E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

DISEÑO E IMPLEMENTACIÓN DE SOFTWARE EN MATLAB PARA LA ADQUISICIÓN, ANÁLISIS Y REPRESENTACIÓN DE DATOS ACÚSTICOS

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Autor: Javier Muruzábal Iturain.
Tutor: Ricardo San Martín Murugarren.
Pamplona, 28 de junio de 2017

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ÍNDICE

RESUMEN

Se propone como objetivo principal la implementación de un software en código abierto ejecutado sobre MATLAB que gestione tanto la adquisición como el manejo de datos acústicos procedentes de medidas y simulaciones realizadas con equipamiento específico del Laboratorio de Acústica. El diseño debe ser modular, de forma que sus prestaciones puedan ser aumentadas o mejoradas. Bajo el motor de *la open source* ITA-Toolbox desarrollada por el *Institute of Technical Acoustics* de la *RTWH Aachen University*, funcionará también a modo de contenedor de futuras aplicaciones ad hoc que se desarrollen en el grupo de Acústica.

Se pretende que se convierta en una herramienta global, es decir, que pueda ser utilizada tanto en actividades avanzadas de investigación como en usos docentes y de prácticas de laboratorio, facilitando la administración de las sesiones habituales de medida y el posterior análisis de los datos obtenidos. Asimismo, deberá contar con tutoriales detallados de cada una de sus especificaciones y una interfaz gráfica eficaz e intuitiva, tanto a nivel de funcionalidad como de usabilidad.

ABSTRACT

The main objective is to implement an open source software executed in MATLAB that manages the acquisition and management of acoustic data from measurements and simulations performed with specific equipment of the Acoustics Laboratory. The design must be modular, so that its performance can be increased or improved. Under the engine of the open source ITA-Toolbox developed by the Technical Acoustic Institute of the University of Aachen RTWH, it will also serve as a container for future ad hoc applications developed in the Acoustics group.

It is intended to become a global tool, that is, it can be used in advanced research activities, as well as in teaching and laboratory practices, facilitating the administration of the usual measurement sessions and subsequent analysis of the data obtained. In addition, you should have detailed tutorials of each of its specifications and an effective and intuitive graphical interface, both in terms of functionality and usability.

PLANIFICACIÓN

- Estudio y análisis de la librería de código abierto (ITA-Toolbox)
    - Estilo de programación
    - Implementación de funciones
    - Sintaxis empleada
- Implementación de aplicaciones adicionales a la librería original
    - Parámetros acústicos
    - Generación de medidas
    - Medida de niveles
    - STI
- Creación de prácticas a modo de tutoriales para indagar en el motor de la ITA y GUIARTE-Toolbox
    - Práctica 1
    - Práctica 2
    - Práctica 3

LISTA DE PALABRAS CLAVE

- Toolbox / Librería
- Open Source / Código abierto
- Software / Programa
- GUI
- MATLAB

Capítulo 1: Introducción

Open Source

Historia

El concepto *open source* surge en los ochenta, época en la que la gran mayoría del software era privado. Surgió la necesidad por parte de los programadores de impulsar un nuevo modelo de software gratuito, el denominado software libre. Se volvía, en cierta medida, a la filosofía de los inicios de la computación, donde los primeros programas informáticos eran desarrollados en un entorno cooperativo, pero que sin embargo tras la aparición de las primeras marcas de ordenadores comenzó a privatizarse.

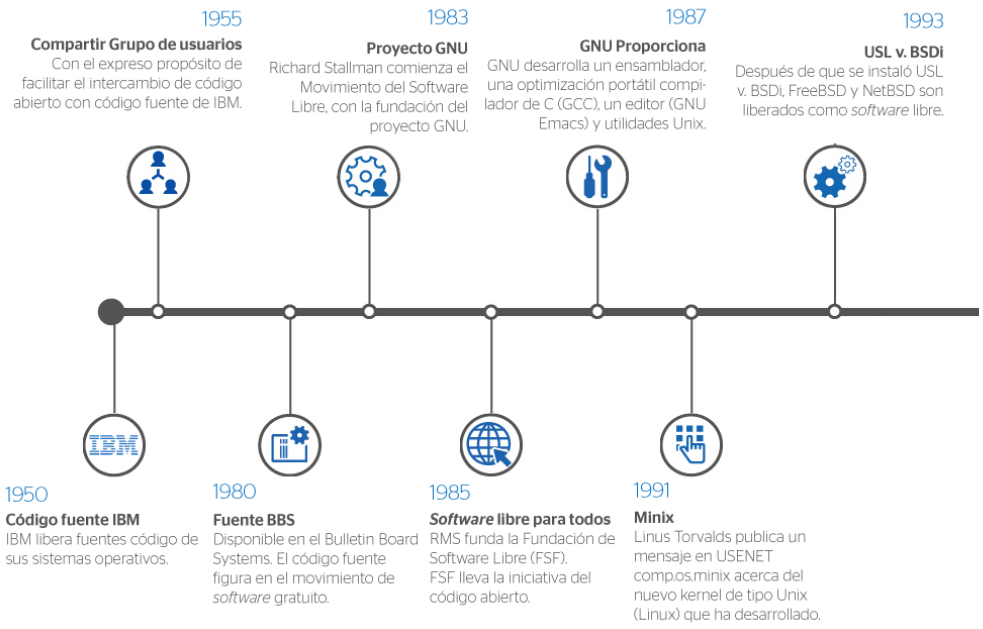

Ilustración 1: Cronología evolución software (1950-1993)

Los primeros proyectos publicados de software *open source* también conocido como código abierto surgieron en los 90, de la mano de varios usuarios de la comunidad de software libre, tan en auge en aquella época, intentando sustituirlo por esta nueva vertiente. Esta evolución implicaba no sólo la posibilidad de emplear el software gratuitamente, nos permitía modificar e incluso redistribuirlo sin reemplazar, en un futuro, las reglas de libertad a nivel de licencias.

Sin embargo, esta nueva modalidad en los inicios no resultó apropiada para algunos programadores debido a que se confundían las ideas de libertad y gratuidad. Diferenciando de este modo los programas de código abierto, que permiten a sus usuarios la libertad de mejorarlos, frente a los programas en los que encontramos el código fuente disponible, pero en los que localizamos restricciones tanto para su uso como modificación.

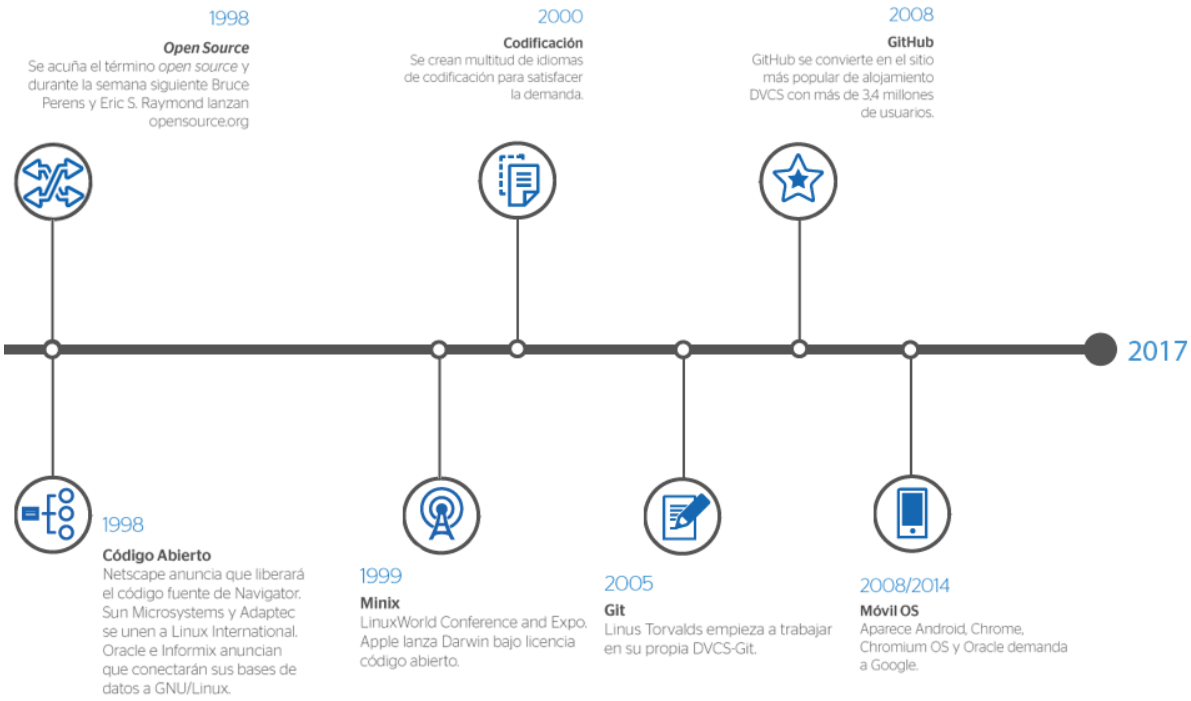

Ilustración 2: Cronología evolución software (1998-2017)

Actualmente, el desarrollo y distribución de software de código abierto está tomando gran protagonismo en el campo de la programación. Entendiendo hoy en día el término software libre no sólo el hecho de adquirir de manera gratuita una aplicación informática, más bien, la posibilidad de modificar el código fuente original sin ningún tipo de restricción a nivel de licencias. Incrementando su atractivo frente al software comercial, de excesivo coste, en el que se oculta su código y se restringen los derechos de licencia.

Plataformas

Con la novedad del *open source* se empezaron a generar en la red portales de intercambio de código. Principalmente destacaremos dos de los portales que por excelencia son conocidos por todos los programadores de hoy en día y una plataforma específica para MATLAB, programa a partir del que realizaremos la programación de nuestro software:

- **GitHub:** es una plataforma colaborativa de desarrollo de software libre, es considerado el mayor portal de desarrollo colaborativo en Internet. El código se almacena públicamente, aunque también se puede hacer de forma privada, creando una cuenta de pago. Puedes cooperar con otros desarrolladores para mejorar su propio software mediante lo que denominan hacer un *fork* y solicitar un *pull*.

Realizar un *fork* significa clonar, generar una copia, de un proyecto en tu cuenta para eliminar posibles bugs o realizar modificaciones que mejoren las funcionalidades del software. Tras realizar las modificaciones se envía un *pull* al dueño del proyecto. Él analizará los cambios realizados y en caso de considerarlos satisfactorios los adjuntará en el proyecto.

Además, incluye una serie de herramientas que fomentan el trabajo en equipo:

- Una wiki donde se explican cada una de las modificaciones realizadas en cada una de las versiones del proyecto.
- Un seguimiento de problemas que permite a los miembros del grupo detallar cualquier error o sugerir nuevas funcionalidades en el software.
- Una herramienta de revisión de código donde podemos añadir en cualquier línea del código comentarios referidos a los posibles cambios o mejoras a implementar.
- Un visor de ramas donde se pueden visualizar los cambios realizados en nuestro proyecto, especificando la fecha en la que ha sido actualizado cada uno de los ficheros.

- **GitLab:** es una plataforma de características similares a GitHub, pero que nos proporcionada la posibilidad de tener repositorios privados ilimitados sin ser poseedores una cuenta de pago.

  Además, integra alguna mejora respecto a sus competidores más cercanos, como es el caso de una interfaz gráfica mejorada en términos de comodidad y dinamismo.

- **Mathworks File Exchange:** *Mathworks* conocida principalmente por el potencial de MATLAB implementa una plataforma de intercambio de proyectos basados todos ellos en el lenguaje empleado en MATLAB. Pese a no tener tanto protagonismo como los repositorios sirve de gran utilidad en la comunidad *Mathworks*.

  Integra las siguientes características:

  - Posibilidad de subir ficheros hasta 20MB.
  - Enlazar el proyecto con repositorios GitHub, File Exchange no los almacenará localmente y será necesario ir a la plataforma Git para descargar el código.
  - Añadir etiquetas y filtros en el caso de que se desee realizar una búsqueda selectiva.

- Capacidad de evaluar y comentar los proyectos con el fin de aportar a otros usuarios reconocimiento en la comunidad.

Herramientas

Con ayuda de estos portales encontramos diversas aplicaciones de interés en el campo de la acústica. A continuación, detallaremos brevemente los programas de código abierto más relevantes:

Análisis de recintos:

- **i-Simpa:** es un software dedicado al modelado 3D de recintos para su análisis acústico. Sus principales tareas son el análisis de la acústica de recintos, el ruido ambiental y el ruido industrial, pero puede ser apto para otras aplicaciones teniendo en cuenta la propagación del sonido en entornos tridimensionales.

  Inicialmente surgió como una herramienta de investigación, pero puede ser una herramienta útil en el ámbito profesional, así como para educación.

- **OpenPSTD:** es una herramienta, orientada principalmente a investigaciones por parte de estudiantes, permitiendo calcular la propagación del sonido en edificios.

  Sin embargo, a diferencia de i-Simpa esta aplicación funciona únicamente en entornos bidimensionales y no cuenta con actualizaciones tan periódicas.

Toolbox MATLAB:

- **Edge Diffraction Toolbox:** esta *toolbox* contiene un conjunto de funciones que calculan la respuesta al impulso para una fuente.

- **ITA-Toolbox:** *toolbox* creada por parte de la universidad de RWTH Aachen. Posee multitud de aplicaciones para solucionar las tareas de post-procesado en el campo de la acústica. Integrando tareas de importación y exportación de datos, así como la posibilidad de diversas representaciones de los datos.

Tras la evaluación de las características de cada una de las herramientas mencionadas anteriormente, se podría decir que el mayor potencial lo encontramos en la última herramienta citada. Asociado a su potencial, es destacable la constante actualización de la *toolbox* y la posible cooperación de cualquier interesado en su desarrollo a través de la plataforma *GitLab*.

Considerando todas estas aptitudes, resultó interesante indagar de manera exhaustiva en todas las posibilidades que esta herramienta integra y siguiendo la filosofía *open source*, realizar alguna modificación propia que ayude tanto al personal docente de la Universidad como a estudiantes.

ITA-Toolbox

Orígenes

La ITA-Toolbox fue desarrollada hace aproximadamente 10 años con el objetivo de suplir las tareas de post-procesado en el campo de la investigación acústica, incluyendo tanto la importación de datos como la representación gráfica en diferentes dominios de esta información.

En los inicios se creó únicamente para el uso académico o educacional hasta que en 2011 las condiciones de uso cambiaron, pasando a ser una herramienta de código abierto con el fin de proporcionar mayor participación en el ámbito de análisis acústico en MATLAB.

Durante el paso de los años, las funcionalidades de la *toolbox* han ido incrementando y se han desarrollado nuevas aplicaciones para tareas de procesado de señal y medición. En el 2012 se lanzó un *kernel* que contenía las funcionalidades básicas y que ha seguido una evolución considerable hasta nuestros días, proporcionándonos actualmente numerosas aplicaciones para el estudio acústico.

Recientemente, la universidad de RWTH Aachen ha decidido subir su proyecto a la plataforma *GitLab* de modo que nos podemos beneficiar de todas las ventajas que una plataforma *Git* implica. Se puede interactuar y visualizar cambios cronológicamente de manera cercana respecto años atrás, incluso reportar bugs que ayuden a sus desarrolladores a mejorar la aplicación.

Concepto y programación orientada a objetos

Las medidas acústicas normalmente son almacenadas en un vector numérico o matriz. Los desarrolladores de la ITA encontraron cierta lógica en almacenar información

extra como puede ser la frecuencia de muestro de la medida realizada, coordenadas, ciertos comentarios que ayuden a referenciar una medida. Programar un contenedor para todos estos datos se puede realizar de manera eficiente a través de la programación orientada a objetos (POO).

Tras diversas pruebas se desarrolló una clase denominada *itaValue* y que actuará como base de almacenamiento de datos en la ITA. Esta clase es capaz de almacenar los datos obtenidos junto con la información del dominio en el que estemos (*itaCoordinates*), es decir, tenemos la posibilidad de realizar transformaciones de coordenadas cartesianas a cilíndricas o esféricas. Simplemente utilizando *.cart* para coordenadas cartesianas o *.sph* para esféricas los datos son convertidos de acuerdo con la representación a realizar. El mismo concepto es extendido a la clase *itaAudio* que almacena audio muestreado de medidas o simulaciones tanto en tiempo como en frecuencia. El dominio temporal siempre está disponible a través de *.time* y para la información en frecuencia ocurre lo mismo con *.freq*. Los puntos a representar de los vectores de tiempo y frecuencia están calculados por *.timeVector* y *.freqVector*, respectivamente. Además, operaciones matemáticas como sumas, restas, multiplicaciones o divisiones están implementadas en estos objetos.

Las clases y objetos son las variables fundamentales sobre las que giran todos los procesos referentes a medidas y representación. Encontramos diferentes tipos clases y objetos en función de los procesos de medida y cálculo a realizar. A continuación, especificaremos en cierta medida las variables mayormente empleadas por la ITA, creadas específicamente para sus implementaciones:

- **itaAudio**: Es considerado el objeto base para la ITA cuando importamos un fichero de audio desde MATLAB, convirtiéndolo mediante la función *ita_read* a un objeto de tipo *itaAudio*. Además de encontrar los correspondientes datos de audio, tanto en el dominio temporal como frecuencial, disponemos de una amplia gama de parámetros como pueden ser el grado de la FFT, la duración de la señal, el número de canales, las coordenadas…

- **itaResult**: Como su nombre indica hace referencia a un objeto contenedor de resultados calculados tras la realización de un proceso como por ejemplo el cálculo del *T30* de una señal.

- **itaMSRecord**: Es una clase de medida que integra además de los respectivos datos tomados todas las opciones de configuración realizadas tales como el tiempo de duración de la grabación e infinidad de características seleccionadas en la tarjeta de sonido como pueden ser la frecuencia de muestreo empleada o los canales por lo que grabamos.

- **itaMSPlayBack**: Esta clase de medida se compone de las características que integra la clase *itaMSRecord* y a su vez implementa la configuración

característica de los canales de salida de nuestra tarjeta de sonido. Recordar que en este procedimiento emitimos una señal y grabamos simultáneamente.

- **itaMSTF**: Esta clase integra las características propias de la clase record y a su vez los parámetros de configuración del barrido en frecuencias a emitir; tipo de *sweep* (lineal o logarítmico), duración… para evaluar la respuesta de una sala.

A partir de estas clases la ITA implementa todas las operaciones de post-procesado que el usuario indique. Destacar que en las clases correspondientes a la realización de medidas es de gran importancia la correspondiente configuración de los parámetros que entran en juego para realizarlas satisfactoriamente.

Funcionalidades

Las funcionalidades básicas se muestran en la figura inferior, se dividen en diferentes grupos siguiendo la idea anteriormente explicada de programación orientada a objetos (POO).

Ilustración 3: Esquema funcionamiento ITA-Toolbox

Describiendo cada uno de los grupos siguiendo un orden coherente en lo que sería un proceso de adquisición de datos y su respectivo análisis:

- **Data IO:** Para la posterior realización del análisis de datos primeramente es necesario la obtención de datos. Por ello, se integran una serie de rutinas que en cierta medida están divididas en dos vertientes:

    - **PortAudio/playrec:** Se emplean para la realización de medidas de ruido de fondo, generación y grabación de barridos en frecuencia (*sweeps*), comunicándonos de esta manera con una amplia gama de tarjetas de sonido compatibles, evitando en cierto modo los comúnmente temidos problemas de latencia tan característicos en estos procesos.

    - **MIDI:** La implementación del MIDI para controlar sus equipos a la hora de realizar medidas.

- **OOP (Programación orientada a objetos):** La ventaja de emplear objetos y ficheros *m-file* tiende a almacenar información adicional a los datos de audio necesarios. De esta forma no es necesario especificar por ejemplo la frecuencia de muestreo a la que hemos realizado la grabación. Todas las funciones, métodos y propiedades de las clases pueden ser accesibles desde la ventana de comandos de MATLAB, con acceso directo tanto a la información en el dominio del tiempo como en el de la frecuencia sin realizar implícitamente ninguna transformada de Fourier.

- **Import & Export:** Estas dos rutinas se resumen en dos funciones denominadas *ita_read* y *ita_write*, capaces de leer y escribir en formatos de datos comunes (*.mat*, *.csv*, *.wav*…). La rutina de importación automáticamente extrae de los ficheros de audios la información relevante y la integra en los objetos de audio (*itaAudio*).

- **Plots & GUI:** Una vez que los datos están almacenados en este tipo de objeto es posible realizar varias rutinas de representación gráfica como por ejemplo; *.plot_time* (representación en tiempo), *.plot_freq* (representación en frecuencia) o *.plot_spectrogram* (espectrograma).

- **Processing:** Tras la respectiva toma de datos es posible realizar un amplio abanico de funcionalidades como pueden convoluciones, filtrados en bandas de octavas, transformadas…

- **Knowledge:** Existe información anexa sobre la utilización de la *toolbox* a través de ayudas en formato HTML como PDF. También se proporcionan una serie de funciones a modo de tutoriales para familiarizarnos con las diversas características que integra la aplicación.

Interfaz

La primera vez que arrancamos la aplicación nos encontramos con una ventana de características similares a las que disponemos en una típica representación gráfica de una señal en MATLAB. Mediante una serie de funciones se ha realizado una adaptación de esta ventana para añadir un menú con todas las posibles implementaciones que somos capaces de ejecutar a través de MATLAB.

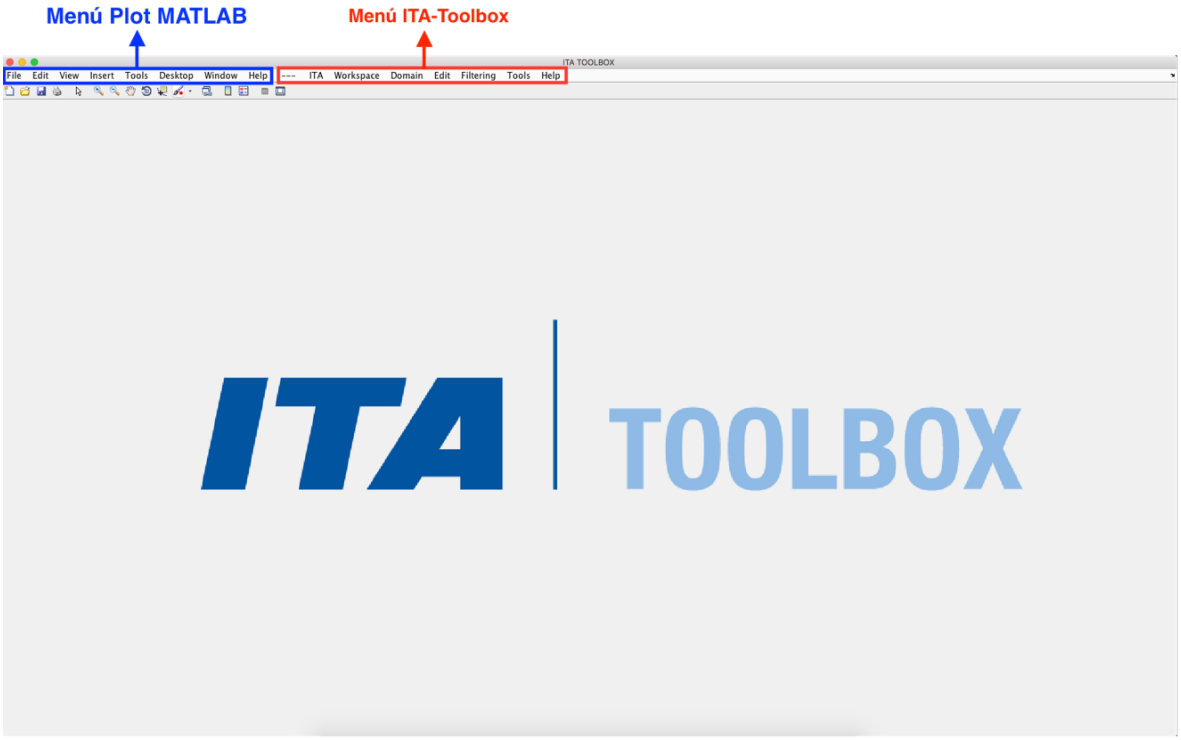

Ilustración 4: Ventana inicio ITA-Toolbox

El menú dispone ordenadamente (izquierda - derecha) de procesos de importación y exportación de datos, representación en diferentes dominios, operación y manipulación de datos y documentación.

En los siguientes apartados se explicará los diagramas se visualiza gráficamente la disposición de cada uno de los submenús de cada uno de ellos.

- **ITA:** En esta sección encontramos todo lo referente a importar y exportar datos en diferentes formatos, así como una extensa configuración en la pestaña de preferencias que en líneas siguientes explicaremos detalladamente pues quizá sea el tema más complejo al que nos enfrentemos con esta aplicación.

  La ventana de preferencias está dividida en cuatro secciones diferencias que concretamente son:

- **General settings:** Como su propio nombre nos indica integra todo lo referente al apartado de ajustes generales como pueden ser información del usuario que emplea la *toolbox*, opciones de visualización o ajustes de carácter general como el rango frecuencial o el número de bandas por octava.

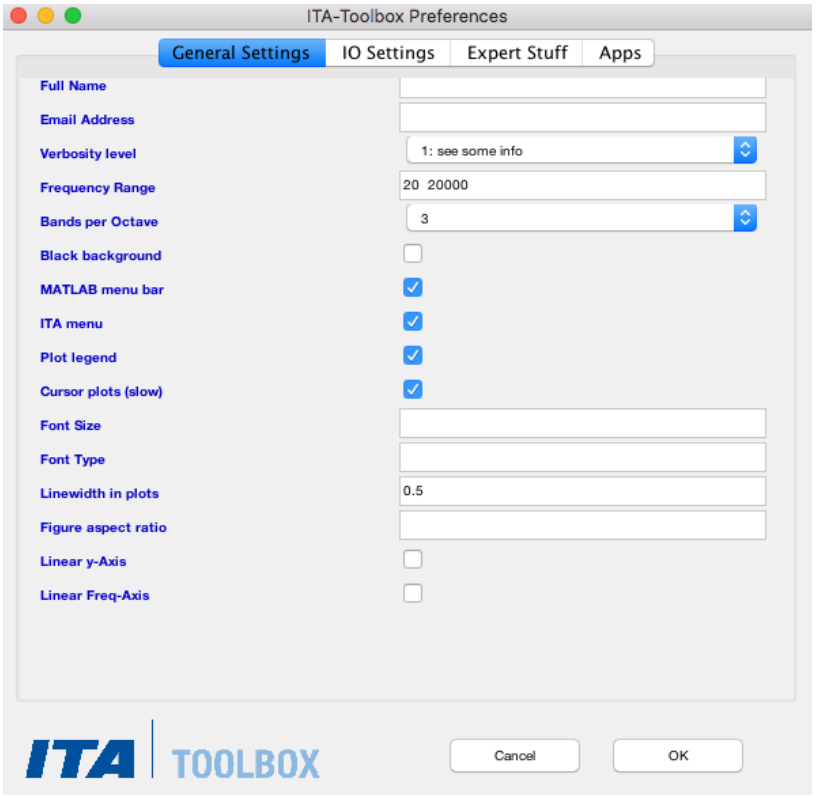

Ilustración 5: Preferencias ITA-Toolbox/General Settings

- **IO Settings:** Sección de configuración que alberga la mayor complejidad para los usuarios que no estén acostumbrados a trabajar con equipos de medida como tarjetas de sonido o interfaces MIDI. Únicamente explicaremos la primera parte de esta ventana puesto como se ha indicado la universidad de RWTH Aachen posee ciertos equipos que para el resto de usuarios de la aplicación no es necesario realizar la configuración de los mismos.

  Tenemos la opción de seleccionar una tarjeta de sonido para los dispositivos de entrada (micrófonos) y otra tarjeta de sonido para las salidas (altavoces) o, por otro lado, seleccionar la misma tarjeta para ambas tareas. Además, podemos elegir la frecuencia de muestreo a emplear por nuestra tarjeta, cuyos rangos de selección variarán en función de su calidad. Es importante que realicemos esta configuración adecuadamente puesto que en un futuro la realización de medidas requerirá de estos ajustes y en caso de una mala configuración los procesos de medida nos mostrarán un error.

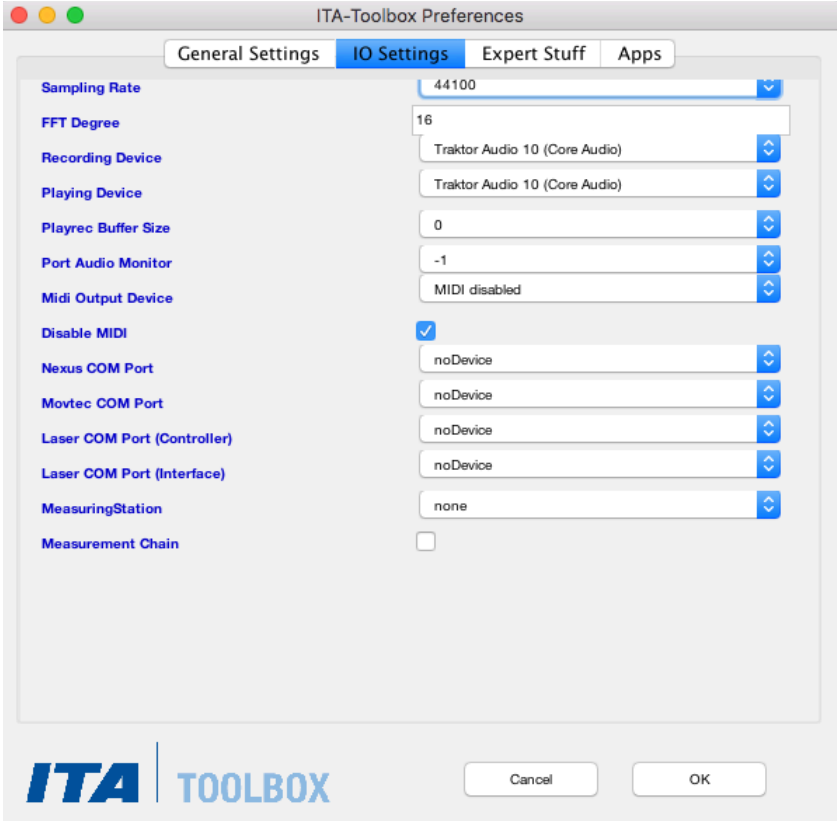

Ilustración 6: Preferencias ITA-Toolbox/IO Settings

- **Expert Stuff:** No nos detendremos en este apartado puesto que no precisa de relevancia para el enfoque que estamos dando al uso de la aplicación.

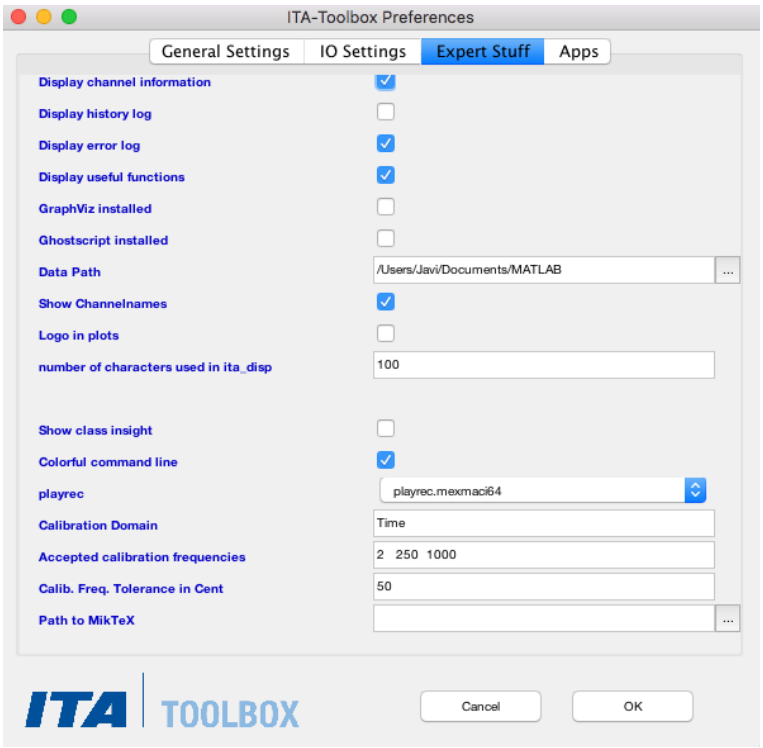

Ilustración 7: Preferencias ITA-Toolbox/Expert Stuff

- **Apps:** Ciertas aplicaciones requieren de una previa configuración y es mediante esta pestaña donde podemos realizar esta tarea.

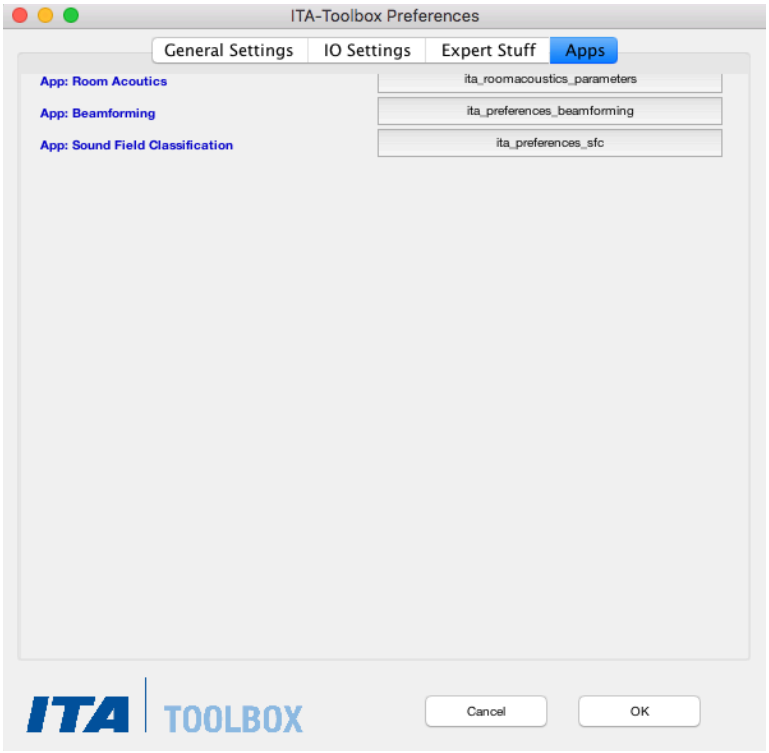

Ilustración 8: Preferencias ITA-Toolbox/Apps

- **Workspace:** El área de trabajo tiene dos funciones, por un lado, se encarga de gestionar los objetos de tipo *itaAudio* que tenemos almacenados y por otro, gestionar en cada momento con qué objeto estamos trabajando. Esta última tarea hay que tenerla muy en cuenta para trabajar con la ITA porque su motor va almacenando dinámicamente todos los cambios que realizamos sobre la *GUI* (interfaz) y no sobre el objeto. Por lo tanto, en caso de querer salvar algún cambio deberemos seleccionar la opción de *Export variable to workspace*, opción situada en este mismo submenú.

- **Domain:** A través de dominio realizaremos las diferentes representaciones de nuestros datos. Es posible su representación tanto en tiempo como en frecuencia, aprovechando la ventaja de los objetos generados y comentados anteriormente en los que se almacenan los datos tanto en el dominio del tiempo como en el de la frecuencia. Se representan en ciertos casos dos gráficos simultáneamente o incluso se realiza una representación a modo de resumen que englobe la mayoría de posibilidades como se ve en la siguiente imagen.

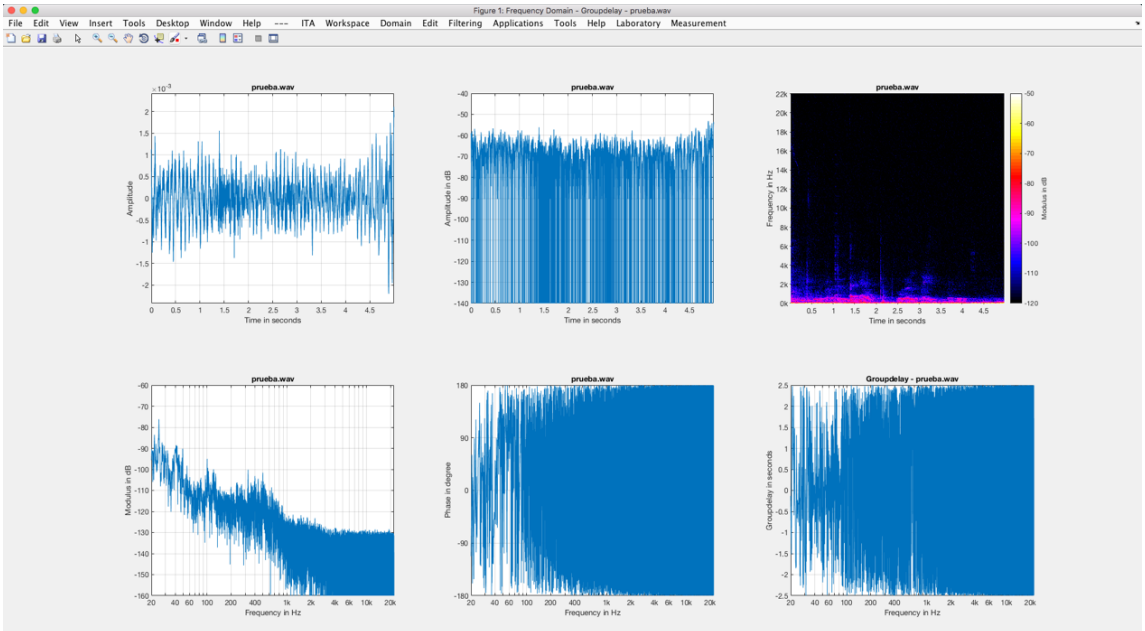

Ilustración 9: Representación gráfica de la señal evaluada en seis dominios diferentes

- **Edit:** Hace referencia a todo lo referente a posibles correcciones y edición de nuestros datos/señales ubicadas en nuestro *workspace*. Podemos amplificar nuestras señales, realizar operaciones entre varias de ellas como pueden ser sumar o restar… Existe la posibilidad también de generar señales *sweep* para su futura utilización en la obtención de la respuesta al impulso de una sala.

- **Filtering:** Disponemos de varios tipos de filtrado entre los que cabe destacar, por su utilización, el filtrado en octavas y el filtrado de octavas en niveles.

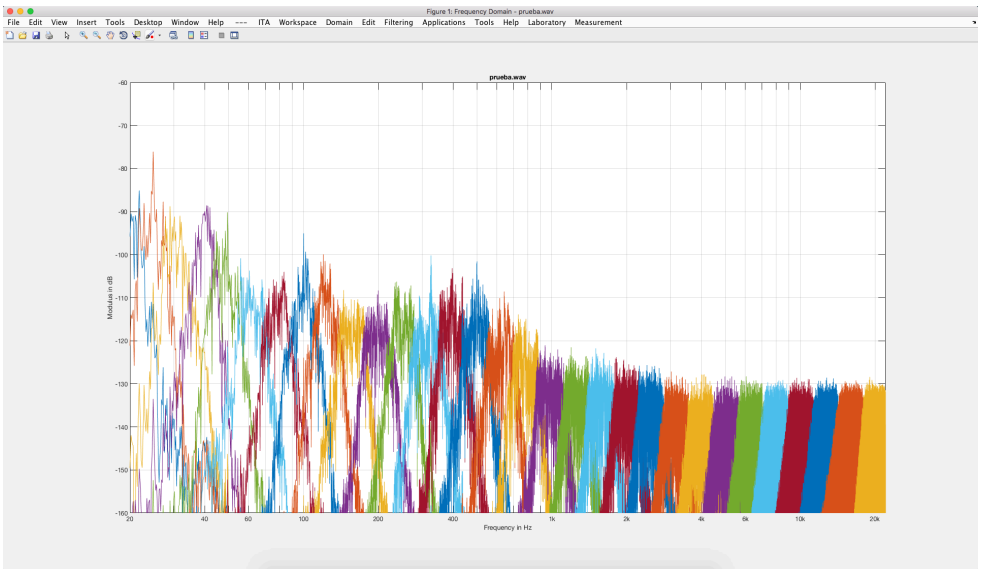

Ilustración 10: Señal filtrada en octavas

- **Tools:** En las herramientas encontramos una serie de funciones que en ciertos casos serán de ayuda, aunque en nuestro caso no han sido empleadas. Pueden

ser de utilidad el generador de tonos frecuenciales o la herramienta de testeo de las entradas y salidas de la interfaz de audio seleccionada en las preferencias.

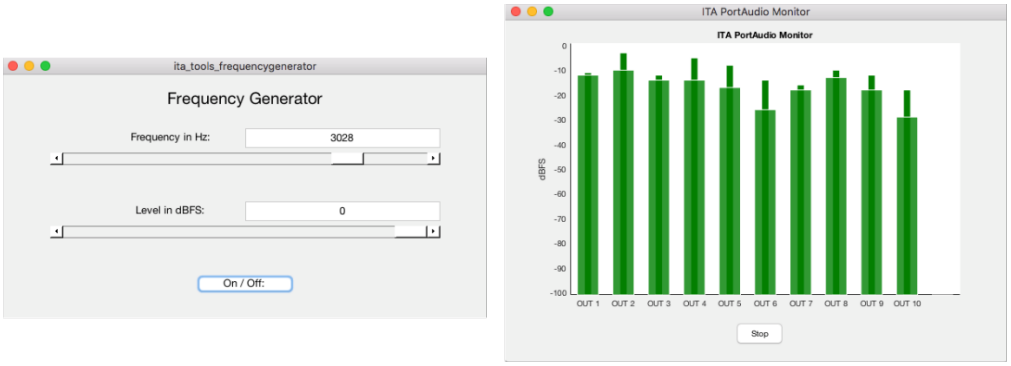

Ilustración 11: Empleo de las herramientas Generador frecuencial y monitor canales de la tarjeta de sonido

- **Help:** A modo de ayuda se ha generado por parte de sus desarrolladores una serie de tutoriales a partir de los que conoceremos con mayor profundidad todas las posibilidades que la *toolbox* posee. Es recomendable la realización de estos tutoriales para conocer y acostumbrarse al funcionamiento característico de la ITA. Por si fuera poco, se incluye un resumen con los diferentes atajos de teclado para acceder de una manera más rápida y eficiente a ciertas funcionalidades.

Reglas de código

Los desarrollados de la *toolbox* han diseñado una serie de normas a emplear por los programadores que quieran cooperar en su evolución. Las clases y funciones dentro de la ITA-Toolbox deben seguir el siguiente estilo:

- *ita_nombre_de_la_funcion*
- *itaNombreDeLaClase*
- Los nombres de las funciones y los métodos solo pueden consistir en nombres en minúsculas y separados con guiones bajos (_). Las definiciones de clase no deben contener espacios, ni caracteres espaciadores entre palabras, pero sin embargo debe empezar cada palabra en mayúscula.
- Tener la seguridad que los nombres de clase, función y variables son representativos de sus tareas y usos.
- Para la creación de nuevas funciones se puede emplear la función *ita_new.m* que creará una plantilla m-file conteniendo las guías para la documentación, *input parsing* y cuerpo de la función.

Alcance y alternativa

Pese a todas las funcionalidades que nos proporciona la ITA-Toolbox encontramos un gran inconveniente en su utilización. Esta aplicación está íntegramente diseñada para su uso con equipos de la universidad de RTWH Aachen, creadores de la ITA. Integrando funciones que son completamente practicables debido a la falta de estos equipos en nuestro entorno de análisis. En relación a lo anterior, implementan una serie de prácticas situadas en el menú *Laboratory* a las que no es posible en todos los casos acceder sin que se nos muestre algún error en MATLAB. Consideramos que este apartado hará referencia a posibles prácticas que los alumnos de esta universidad realicen en algún momento durante su grado universitario.

Es por ello que hemos considerado conveniente realizar, en cierta medida, un parche anexo a la ITA-Toolbox de tal forma que podamos realizar otro tipo de tareas con una amplia gama de equipos, menos específicos, sin necesidad de manipular preferencias de equipos que no sean comúnmente utilizados. Personalizando la aplicación con el objetivo de beneficiar tanto al personal docente, como herramienta de ayuda a la investigación, como a los alumnos para su correcto aprendizaje en el campo de la acústica.

GUIARTE

Descripción

GUIARTE (*Grafical User Interface for Accoustic Rsearch and TEaching*) consiste en una capa personalizada para obtener, representar y analizar datos acústicos sin perder el potencial implementado en el motor de la ITA y la comunicación en ambos sentidos entre las *toolboxes*. Se propone como una solución a los excesivos parámetros de configuración que encontramos en la versión original, permitiendo de este modo realizar procesos y análisis acústicos prescindiendo de ciertos conocimientos tecnológicos ajenos a la acústica.

Además, se han implementado una serie de herramientas interesantes en el campo de la acústica como pueden ser la importación de datos en formatos característicos de otros softwares (*WinMLS* por ejemplo) o exportación de los parámetros acústicos evaluados en hojas de cálculo. La integración de estas funcionalidades sigue el mismo esquema en cuanto a directorios que poseemos en la ITA, separando las aplicaciones del respectivo *kernel*, en el que encontramos el motor de nuestro programa. De esta forma conseguimos que futuras versiones de la ITA se integren sin ningún tipo de incompatibilidad.

Guía de uso

A continuación, se detallarán las características generales que un usuario necesitará manejar para realizar sencillas implementaciones en la GUIARTE-Toolbox, explicando las similitudes y las diferencias con respecto a la ITA.

Cuando ejecutamos el comando *guiarte_toolbox_gui* nos aparece una ventana típicamente similar con un menú propio incrustado al lado del menú característico y anteriormente comentado de la ITA, siguiendo nuestras preferencias. Es importante insistir en su total integridad con las funcionalidades de la ITA, de tal forma que existe comunicación bidireccional entre ambas herramientas.

Ilustración 12: Ventana Principal GUIARTE-Toolbox

Instalación

Para su instalación se ha realizado un proceso similar al realizado por parte de la ITA. Para ello simplemente debemos copiar la carpeta GUIARTE-Toolbox dentro de nuestro directorio donde tenemos ubicado nuestra carpeta de la ITA-Toolbox. Una vez realizado este proceso simplemente debemos ejecutar el script denominado *upna_toolbox_setup.m* (ubicado en la carpeta *Setup*) y automáticamente se realizará la instalación de la librería mostrándose un mensaje en el caso de que la instalación se realice satisfactoriamente.

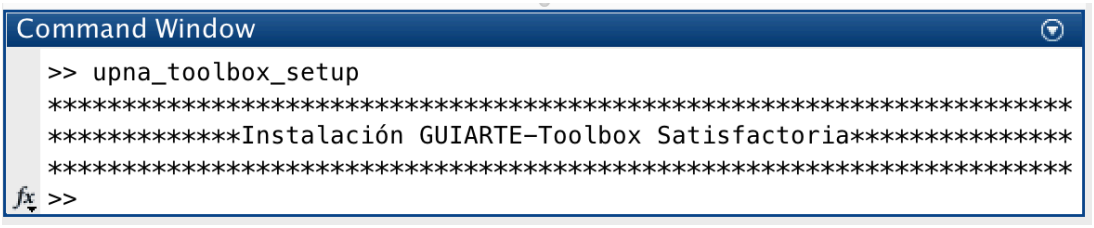

Ilustración 13: Mensaje de instalación satisfactoria en Command Window

En el proceso de instalación se añaden al *Path* nuestras propias funcionalidades y se modifica ligeramente alguna función propia de la ITA.

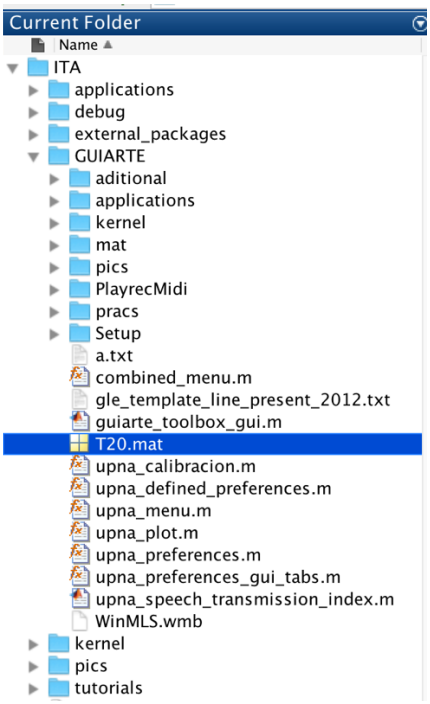

Ilustración 14: Path con ambas librerías y su disposición

Una vez finalizada la instalación será necesario ejecutar el comando *guiarte_toolbox_gui* en el *Command Window* para arrancar nuestra aplicación.

En cualquier caso, es posible eliminar nuestra *toolbox* y recuperar plenamente todas las funcionalidades de la ITA. El proceso sería a la inversa que con la instalación. Se realizará mediante el script *upna_toolbox_uninstal.m* también ubicado en la carpeta *Setup*.

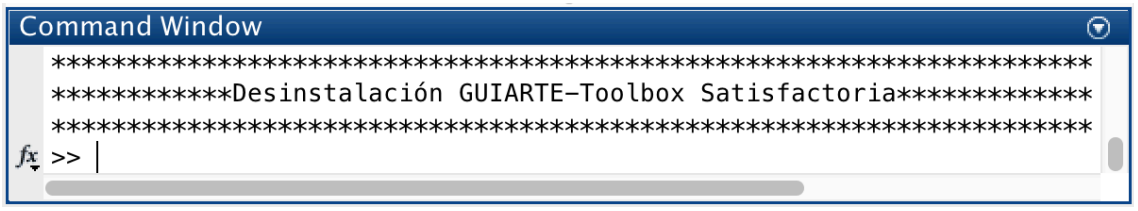

Ilustración 15: Mensaje de desinstalación satisfactoria en Command Window

UPNA

En el apartado UPNA se han modificado ligeramente algunas de sus entradas proporcionando algunas características que ayuden la comunicación con otros programas empleados en la investigación acústica y una simplificación de las preferencias a manipular por parte del usuario.

- **Read:** nos ofrece la posibilidad de importar datos en formato *.wav* (típicamente utilizado en grabaciones) y en *.wmb*, característico este último del programa *WinMLS* comúnmente empleado en la Universidad para la realización de medidas.

- **Write:** características idénticas a las empleadas por parte del motor ITA. Nos exporta el objeto con el que estemos trabajando al formato anteriormente comentado *itaValue*.

- **Preferences:** en el apartado de las preferencias principalmente se ha realizado un análisis de las configuraciones básicas y necesarias que se podían manipular sin generar ningún tipo de problema, que el usuario de nivel medio no pueda solventarlas. Siguiendo con este patrón se ha realizado una comunicación en ambos sentidos entre las preferencias de la ITA y las propias de nuestra *toolbox*.

  Como se puede ver en las siguientes ilustraciones se ha simplificado considerablemente el apartado ajustes suprimiendo todas las preferencias referentes a equipos específicos empleados por parte de la universidad de RWTH Aachen.

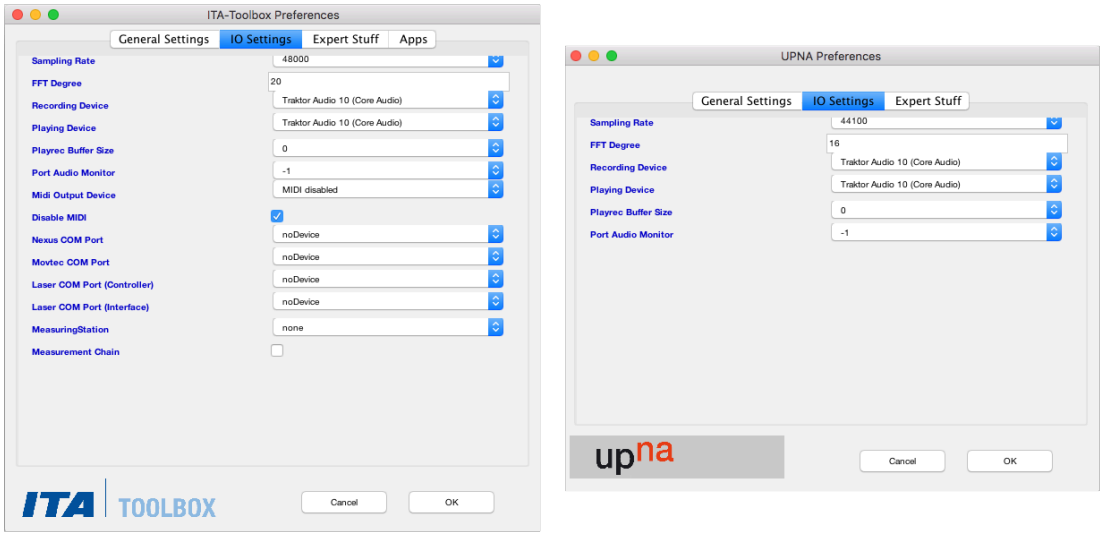

Ilustración 16: Diferencias entre preferencias de la ITA-Toolbox y la GUIARTE-Toolbox apartado IO Settings

- **Save this plot:** proporciona la capacidad de exportar un gráfico en formato imagen para su posterior evaluación.

Applications

Este submenú, por el momento, se describirá brevemente debido a que se considera de interés detallarlo en próximas secciones para conocer detalladamente todo el potencial que es capaz de realizar cada una de las aplicaciones implementadas. También se explicarán los procesos a seguir para ejecutar satisfactoriamente cada una de estas aplicaciones.

Las aplicaciones que se han implementado son:

- **Levels:** Medida de niveles, dividido por bandas, a partir de una señal grabada en el mismo procedimiento.
- **Room Acoustics**
    - **Room Acoustic:** Análisis del parámetro acústico seleccionado y representación de este en bandas y frecuencia.
    - **Room Acoustic Default Parameters:** Selección de los parámetros por defecto que nuestra *toolbox* calculará.
    - **Room Acoustic Export Parameters:** Exportación de los parámetros a una hoja de cálculo para su posterior evaluación.
- **STI:** medida de la inteligibilidad

Pracs

La sección de prácticas albergará una serie de funciones a modo de ejemplos sobre cómo realizar algunas implementaciones para aplicaciones futuras, sin olvidar la utilización de estas prácticas como recurso académico para la comprensión de los conocimientos académicos impartidos. Constará de las siguientes prácticas:

- **Prac1:** Evaluación del tiempo de reverberación (T20).
- **Prac2:** Evaluación de la respuesta frecuencial de auriculares.
- **Prac3:** Evaluación del ruido a través de cabeza binaural.

Measurement

El apartado de medida pese a ser una aplicación se ha considerado que requiere de mayor protagonismo en el menú situándolo externamente a la sección aplicaciones.

Al igual que las aplicaciones, se explicará en posteriores secciones de una manera más detallada, simplemente indicaremos que es posible la realización de tres tipos de medidas; grabación, emisión de un sonido precargado y grabación simultánea y medidas de respuesta de transferencia a través de impulsos a partir de la generación de *sweeps*

El submenú *Measurement* estaría compuesto de la siguiente manera:

- **New Measurement Setup:** Realización de una nueva medida entre las que podemos elegir; *record*, *playback & record* y *transferfunction*.
- **Choose Measurement:** Selección de la medida a modificar o lanzar en el caso de que tengamos varias de ellas creadas.
- **Edit Mesaurement:** Posibilidad de editar los parámetros de configuración de una medida ya configurada.
- **Load Measurement Prefs:** Carga de una configuración de medida previamente guardada.
- **Save Measurement Prefs:** Exportar los parámetros de configuración de una medida realizada.
- **Calibrate Measurement:** Calibración de los equipos. En caso de no existir calibración previa a lanzar una medida se nos avisará con un mensaje.
- **Run Measurement:** Ejecuta la medida previa configuración de la misma.

Capítulo 2: Generación de aplicaciones y protocolos de implementación

Introducción

Siguiendo la filosofía *open source* y la extensión de prestaciones de la *toolbox* es necesario explicar la estructura que deberán seguir las posibles aplicaciones futuras a implementar.

Además, se describirá detalladamente mediante la utilización de ejemplos la ejecución de cada una de las aplicaciones realizadas, mostrando el potencial de nuestra *toolbox*. Dividiremos la explicación en diferentes sub-apartados, en los cuales cada uno de ellos contendrá todas las características referentes a la aplicación correspondiente.

Estructura de la aplicación

Previamente a la programación de nuevas aplicaciones es necesario tener en cuenta la estructura que deberán seguir a nivel de directorios, tal y como se visualiza en la siguiente imagen:

Ilustración 17: Estructura carpetas creación de aplicaciones

Se creará una carpeta con el nombre de la aplicación y dentro de esta carpeta se creará una nueva denominada *NombreAplicacionCallback, siguiendo la misma sintaxis de mayúsculas y minúsculas*. Esta última carpeta contendrá dos tipos de funciones:

- ***upna_guimenuentries_nombreapliacacion*:** indica los apartados que contendrá el menú y la respectiva llamada a las funciones a ejecutar para cada uno de los procedimientos.

```
function upnamenu = upna_guimenuentries_measurement()
% GUIARTE_GUIMENUENTRIES_MEASUREMENT - Defines Measurement Menu entries

% <GUIARTE-Toolbox>

idx = 1;
upnamenu{idx}.type   = 'submenu';
upnamenu{idx}.text   = 'Measurement';
upnamenu{idx}.parent = '';

idx = idx+1;
upnamenu{idx}.type   = 'function';
upnamenu{idx}.text   = 'New Measurement Setup';
upnamenu{idx}.parent = 'Measurement';
%upnamenu{idx}.separator = true;

idx = idx+1;
upnamenu{idx}.type   = 'function'; % evtl Varlist
upnamenu{idx}.text   = 'Choose Measurement';
upnamenu{idx}.parent = 'Measurement';
```

- ***upna_menucallback_nombrefuncion*:** esta función tiene que ser definida con el mismo nombre que se le ha asignado en el *guimenuentries*, pero sin espacios. Por ejemplo, para el caso de *New Measurement Setup* el nombre del script sería; *upna_menucallback_NewMeasurementSetup*. Una vez se ha asignado el nombre al script de manera satisfactoria procederemos a implementar el código de dicha función o incluso la posibilidad de llamar a otras funciones internamente.

```
function upna_menucallback_NewMeasurementSetup(hObject, eventdata) %#ok<INUSD>

% <GUIARTE-Toolbox>

% gets variables from Workspace
count = ita_getfrombase('count');

if isempty(count)
    count = 1;
else
    count = count +1;
end

filename = ['MS' num2str(count)];

MS = upna_measurement();

%set the variables into the workspace
ita_setinbase(filename, MS);
ita_setinbase('count', count);
ita_setinbase('actMS', filename);

end
```

- **Funciones adicionales:** las funciones adicionales necesarias a implementar serán almacenadas en la raíz de la carpeta de la aplicación o en el caso de contar

con un número considerable de funciones almacenarlas en otra carpeta denominada *aditional*.

Finalmente, indicar que para una correcta coherencia con la ITA la asignación de variables, métodos, funciones… será en inglés.

Creación de GUIs

Un aspecto muy a tener en cuenta a la hora de crear una aplicación es la posible interacción que el usuario tendrá con la herramienta. Para ello y siguiendo los criterios implementados por parte de la ITA indicaremos cómo realizar tanto ventanas que servirán al usuario como entrada de las diferentes opciones proporcionadas en cada proceso, así como la generación del proceso de actualización de la ventana principal a la hora de realizar representaciones gráficas.

Ventanas dinámicas

La creación de ventanas dinámicas será el elemento inicial en base al que se realizarán las nuevas aplicaciones, permitiendo al usuario interacción con la herramienta. Para ello a continuación se especificarán los diferentes tipos de entrada disponibles y el uso concreto de cada uno de ellos.

```
ele = 1;
pList{ele}.description = 'FFT Degree';
pList{ele}.helptext    = 'This is the itaAudio Object for amplification or attenuation';
pList{ele}.datatype    = 'int';
pList{ele}.default     = '18';
```

La generación de *inputs* en la ventana se hará mediante un listado (vector) de elementos y correspondiendo a cada elemento las siguientes características:

- **Descripción:** Hará referencia a la entrada correspondiente
- **Helptext:** Texto de ayuda en el caso de desconocer a qué hace referencia esa entrada
- **Datatype:** La característica más importante de cada uno de los elementos del listado. Existen los siguientes tipos:
    - Int: la variable de entrada será un entero.
    - Char: la variable de entrada será una cadena de texto

- Simple_button: hará llamada a una función a través de un callback.

```
ele = numel(pList)+1;
pList{ele}.description = 'Preferences';
pList{ele}.helptext    = 'Call upna_preferences()';
pList{ele}.datatype    = 'simple_button';
pList{ele}.default     = '';
pList{ele}.callback     = 'upna_preferences();';
```

- Int_result_button: también llamada a una función, pero en este caso devuelve un listado de enteros. Un ejemplo claro de este procedimiento puede ser la elección de las entradas seleccionadas en una tarjeta de sonido.

```
ele = numel(pList)+1;
pList{ele}.description = 'Input Channels';
pList{ele}.helptext    = 'Call upna_preferences()';
pList{ele}.datatype    = 'int_result_button';
pList{ele}.default     = '1 2';
if isempty(pList{ele}.default)
    pList{ele}. default = [1 2];
end;
pList{ele}.callback = 'ita_channelselect_gui([$$],[],''onlyinput'')';
argList = [argList {'inputChannels'}];
```

- Boolean: mediante la utilización de un booleano (dos estados 1 o 0) podemos marcar ciertas casillas para seleccionar opciones como por ejemplo si al realizar una medida se desea que se almacene en un objeto.

```
pList{1}.description = 'Save Measurement';
pList{1}.helptext    = '';
pList{1}.text        = 'Tick for Save Measurement';
pList{1}.datatype    = 'bool';
pList{1}.default    = 1;
```

Con estos tipos de elementos seríamos capaces de construir cualquier tipo de entrada para la creación de una ventana, pero llegados a este punto necesitamos una interfaz a través de la cual mostrar las entradas, es decir, un contener en el que visualizarlas.

```
pList = upna_parametric_GUI(pList,'Titulo Correspondiente');
```

Mediante la función *upna_parametric_GUI* le pasamos el listado de elementos de entrada y esta función automáticamente muestra una ventana con dimensiones acordes al número de elementos del listado.

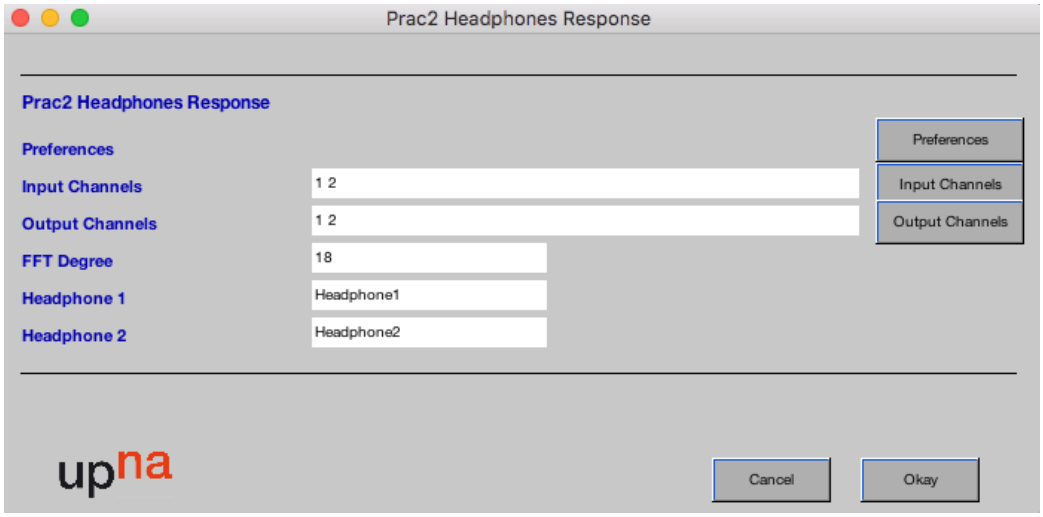

Ilustración 18: Ejemplo de ventana creada a través de *upna_parametric_GUI*

Update GUIs

La idea de emplear como herramienta de trabajo siempre la ventana principal nos sugiere la creación de funciones capaces de actualizar la GUI principal. Dependiendo del tipo de representación que consideremos oportuna será necesario implementar una función adicional de este tipo. Mediante el siguiente ejemplo se pretende comprender la filosofía de estas *update GUIs*.

```
function upna_guisupport_updateGUI_freq(fgh, h1, h2)

% plot audio in figure
currentDomain = getappdata(fgh,'ita_domain');
data =  getappdata(fgh);

clf(fgh, 'reset')

if isempty(currentDomain)
    error('')
end

subplot(2,1,1);
    ita_plot_freq(h1,'figure_handle',fgh);
    title(h1.comment);
    legend('L signal','R siganl');


subplot(2,1,2);
    ita_plot_freq(h2,'figure_handle',fgh);
    title(h2.comment);
    legend('L signal','R siganl');

combined_menu('handle',fgh);

end
```

El elemento de entrada considerar es el denominado *fgh* y que hace referencia a las ventanas de MATLAB denominadas figure. Una vez realizada la llamada al elemento *figure* procederemos a implementar las representaciones gráficas necesarias mediante

la instrucción *subplot(m, n, x)* con m (número de filas del *plot*), n (número de columnas del *plot*) y x índice del *plot*.

Tras la implementación de las representaciones gráficas llamaremos a *combined_menu()* para que nos vuelva a integrar la combinación del menú de la ITA y nuestro propio menú de nuevo en la ventana principal.

Aclarada la idea de cómo integrar las funciones referentes a la interfaz gráfica de nuestra herramienta vamos a dar paso a la explicación de las aplicaciones implementadas, haciendo referencia en los procedimientos relevantes seguidos para conseguir la finalidad deseada.

Aplicaciones

Room Acoustics

Room Acoustics hace referencia a la aplicación que engloba todos los parámetros implementados. Podemos visualizar de manera gráfica o numérica los valores obtenidos para cierto parámetro acústico, incluso exportar los parámetros convenientes a formato Excel para su posterior análisis.

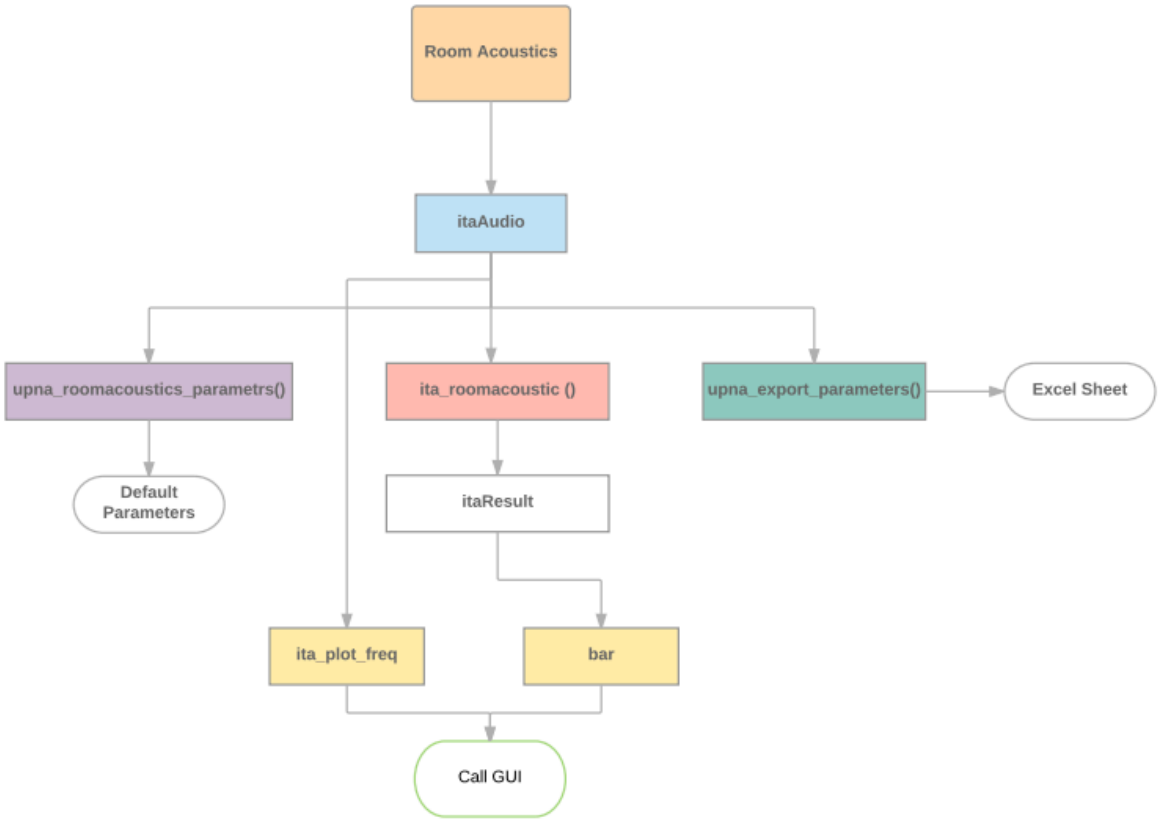

Diagrama de flujo 1: Aplicación Room Acoustics

Es importante para el correcto funcionamiento de la aplicación cargar previamente a través del menú UPNA el correspondiente fichero de audio sobre el que queremos calcular los diversos parámetros. En caso de no realizar esta operación previa será necesario reiniciar la *toolbox* debido a los errores ocasionados.

Room Acoustic

Una vez realizado el proceso de importación de nuestro fichero de audio procedemos a ejecutar el método *Room Acoustic*. A través de la ventana que se adjunta el usuario se encarga de seleccionar el parámetro a evaluar para cierto objeto *itaAudio*.

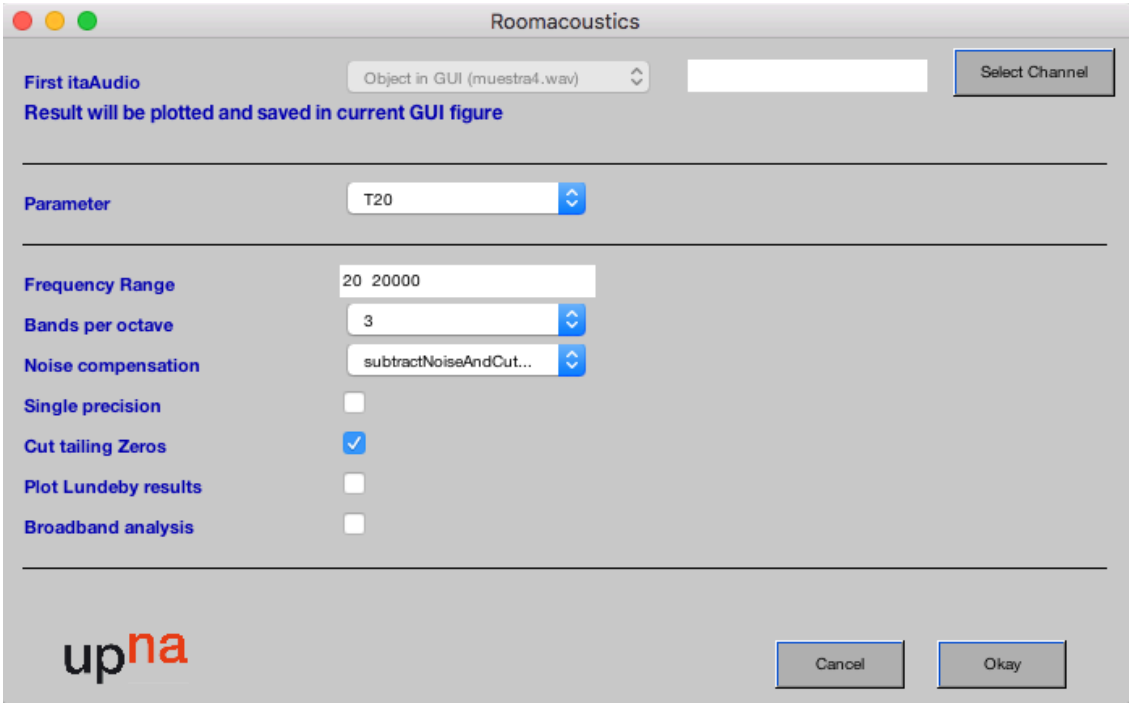

Ilustración 19: Ventana Room Acoustics

Como se puede observar la ventana está dividida en tres secciones: *itaAudio* seleccionado, parámetro acústico a evaluar y una serie de preferencias a tener en cuenta como pueden ser las bandas por octava (octavas o tercios) y los diferentes modos de compensación de ruido integrados por el motor ITA. Una vez realizada la correcta selección de las características aceptaremos mediante el botón *Okay* situado en la zona inferior derecha y la herramienta procederá al cálculo y posterior representación gráfica.

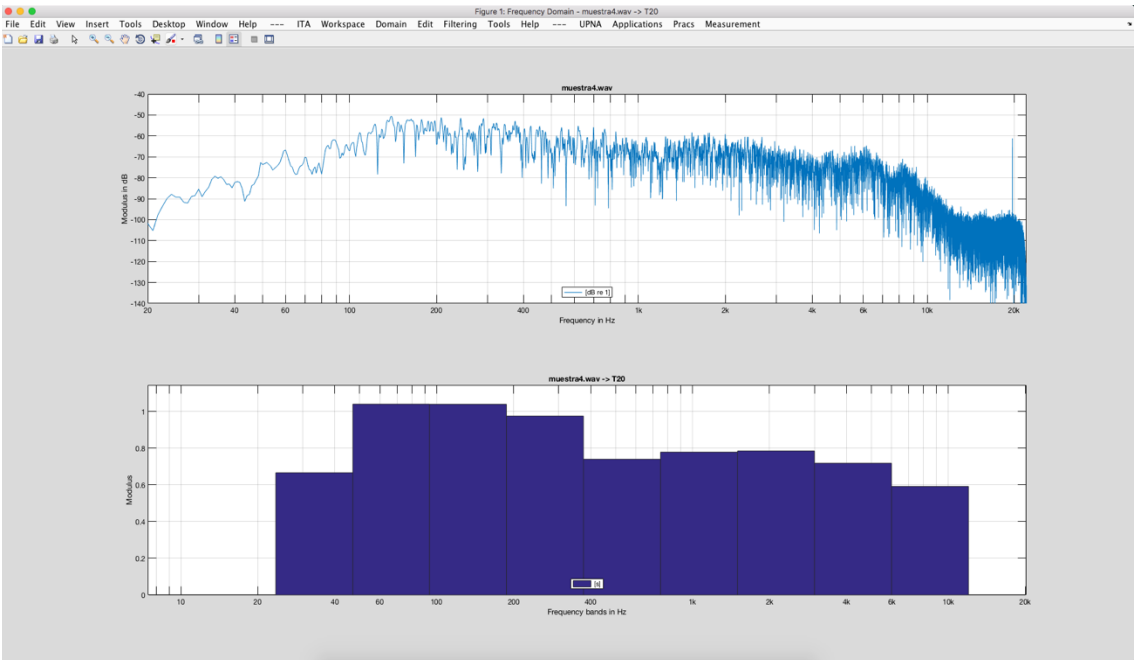

Ilustración 20: GUI dividida en representación en frecuencia y representación del T20

Como se puede observar se ha dividido la ventana en dos gráficos (representación de la señal en frecuencia y representación en bandas del parámetro acústico evaluado) mediante la instrucción *subplot*, mejorando en este apartado la funcionalidad que permitía la ITA y su vez asemejándose a la representación realizada por el software *WinMLS*. Para la imagen adjunta (ilustración 20) han sido seleccionadas las características mostradas anteriormente en la ilustración 19.

El código referente a la representación gráfica, dividiendo la ventana en dos gráficos se ha realizado:

```
function upna_guisupport_updateGUI_dual(fgh)
% plot audio in figure

currentDomain = getappdata(fgh,'ita_domain');
data =  getappdata(fgh,'audioObj');


clf(fgh, 'reset')

if isempty(currentDomain)
    error('')
    currentDomain = data.domain;
end

logic=isempty(evalin('base','aux'));
```

```
if logic == 1
    fig = evalin('base','backup');
    assignin('base','aux',fig);
else
    fig = evalin('base','aux');
end

subplot(2,1,1);
    ita_plot_freq(evalin('base',fig),'figure_handle',fgh);
subplot(2,1,2);
    bar(data,'figure_handle',fgh);

combined_menu('handle',fgh,'type',data);


end
```

Aprovechándonos de las funciones de representación gráfica implementadas por la ITA dividimos la ventana en dos y llamamos a cada una de las funciones correspondientes. En este caso concreto hacemos referencia a *ita_plot_freq* (para representación en frecuencia) y a *bar* (representación en forma de barras de los tiempos de reverberación para cada banda).

Otro aspecto importante a tener en cuenta es el denominado *figure_handle* y *fgh*. Nuestro propósito es siempre trabajar en la misma ventana de la aplicación para ello lo que debemos realizar es una constante actualización del entorno de trabajo. A través de *figure_handle* y *fgh* conseguimos esto.

Room Acoustic Default Parameters

El apartado *Room Acoustic Default Parameters* hace mención a los parámetros que por defecto nuestra *toolbox* calculará de manera previa para su posterior representación. A través de esta implementación conseguimos menores tiempos ejecución, permitiendo al usuario interactuar con la herramienta de forma fluida. El tiempo de ejecución para la representación de un parámetro previamente calculado es relativamente pequeño.

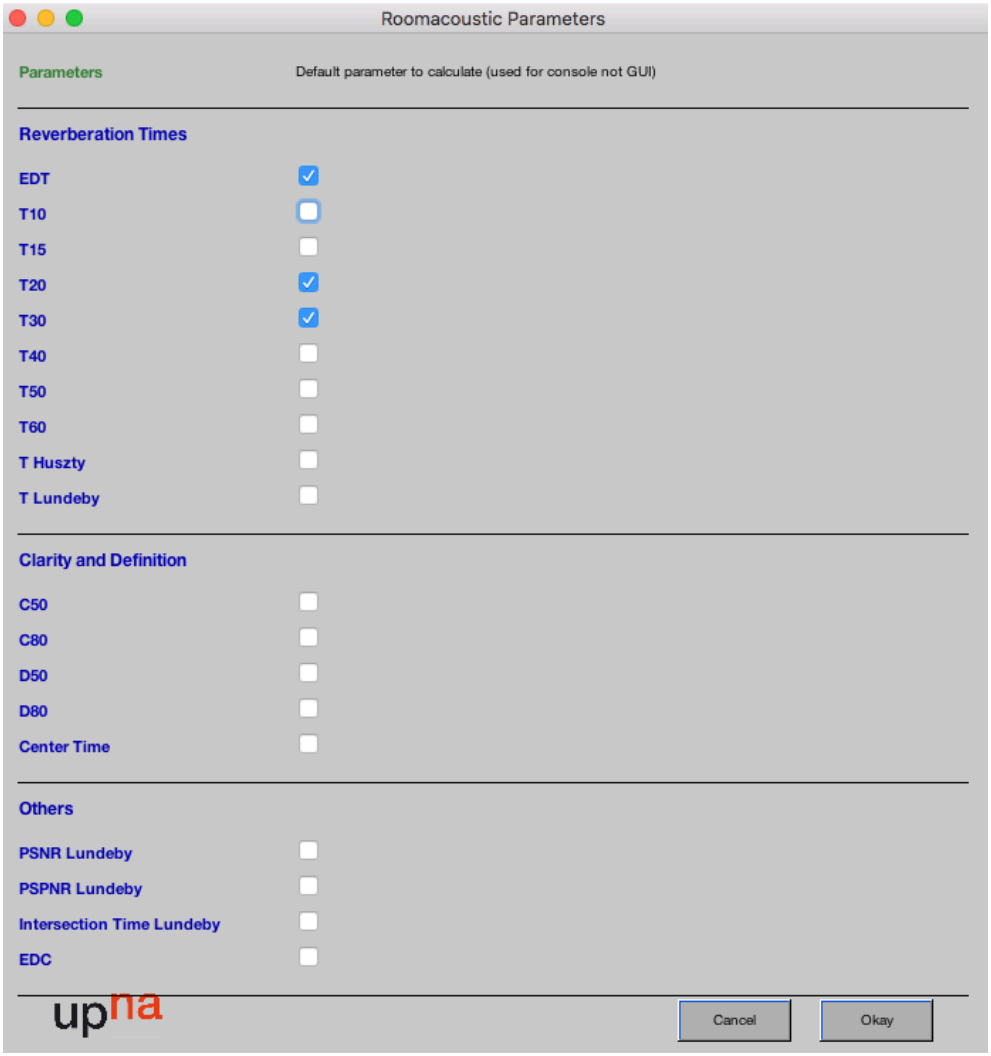

Ilustración 21: Ventana Room Acoustic Default Parameters

Los parámetros acústicos implementados se han dividido en tres secciones: tiempos de reverberación, claridad y definición y otros.

Room Acoustic Export Parameters

Con objeto de almacenar para posteriores cálculos los parámetros correspondientes a una señal de audio, se ha programado un script que almacena en formato Excel todos los parámetros acústicos por tercios de octava. Simplemente y como se muestra en la ventana inferior tendremos que indicar el directorio donde exportar el fichero *.xlsx* y el nombre que le asignaremos a tal fichero.

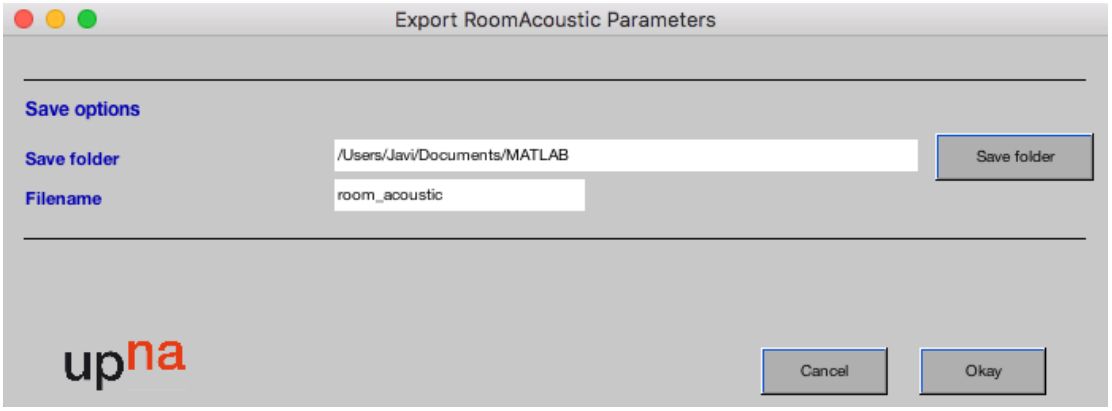

Ilustración 22: Ventana Room Acoustic Export Parameters

El código empleado para la exportación de estos parámetros es el siguiente:

```
if ismac
    directory = [folder '/' filename];
elseif ispc
    directory = [folder '\' filename];
else
    disp('Not supported')
end

existencia = exist(directory);
if (existencia == 0)
    writetable(T,filename,'FileType','spreadsheet')
else
    choice = questdlg('Would you like to replace the file?', ...
    'Replace Matlab file', ...
    'Yes','No','No');
% Handle response
    switch choice
        case 'Yes'
            disp(['Fichero reemplazado'])
            writetable(T,filename,'FileType','spreadsheet')
        case 'No'
            disp(['Guardado cancelado'])
            upna_export_parameters;
    end

end
```

Las variables *folder* y *filename* corresponden a los valores introducidos en la ventana *export parameters*. Primeramente, definimos en qué sistema operativo estamos trabajando (Mac OS o Windows) se nos generará un fichero con el nombre y directorio introducidos por el usuario acorde al S.O. Inmediatamente, se comprobará la posible existencia de este fichero, de tal forma que evitemos los típicos errores de reemplazo de ficheros perdiendo la información anteriormente almacenada en el correspondiente Excel. En caso de existir el fichero *.xlsx* en el mismo directorio y con el mismo nombre nos aparecerá una ventana emergente con la posibilidad de reemplazarlo o en caso contrario modificar el nombre del archivo, tal y como se ve en las líneas de código posteriores a la comprobación del sistema operativo.

Measurement

En el apartado de *Measurement* se podrán realizar diferentes tipos de medidas en función de nuestro propósito. A continuación, se detallará cada una de las entradas de este submenú, explicando los pasos a seguir a la hora de realizar una nueva medida. Como advertencia indicar que es fundamental calibrar el sistema de medida previamente a la ejecución de una medida, procedimiento que se detallará en posteriores líneas.

Diagrama de flujo 2: Aplicación de realización de medidas

New Measurement Setup

Primeramente, al seleccionar una nueva medida nos aparecerá una ventana con las posibles variantes de medidas que concretamente se conforman en grabar, emitir un sonido y grabar simultáneamente o calcular la función de transferencia de una sala (mediante la emisión de *sweeps*).

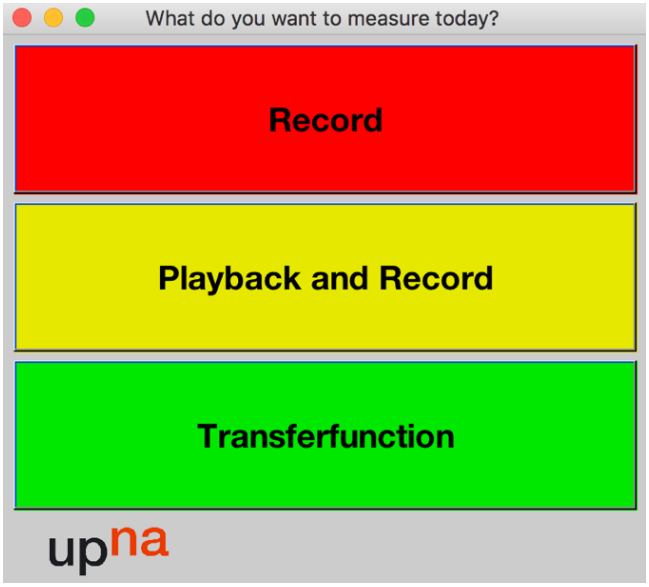

Ilustración 23: Ventana de selección de nueva medida

Una vez seleccionado el tipo de medida a ejecutar podemos modificar la configuración por defecto que se nos presenta en una ventana muy similar a la que se muestra a continuación.

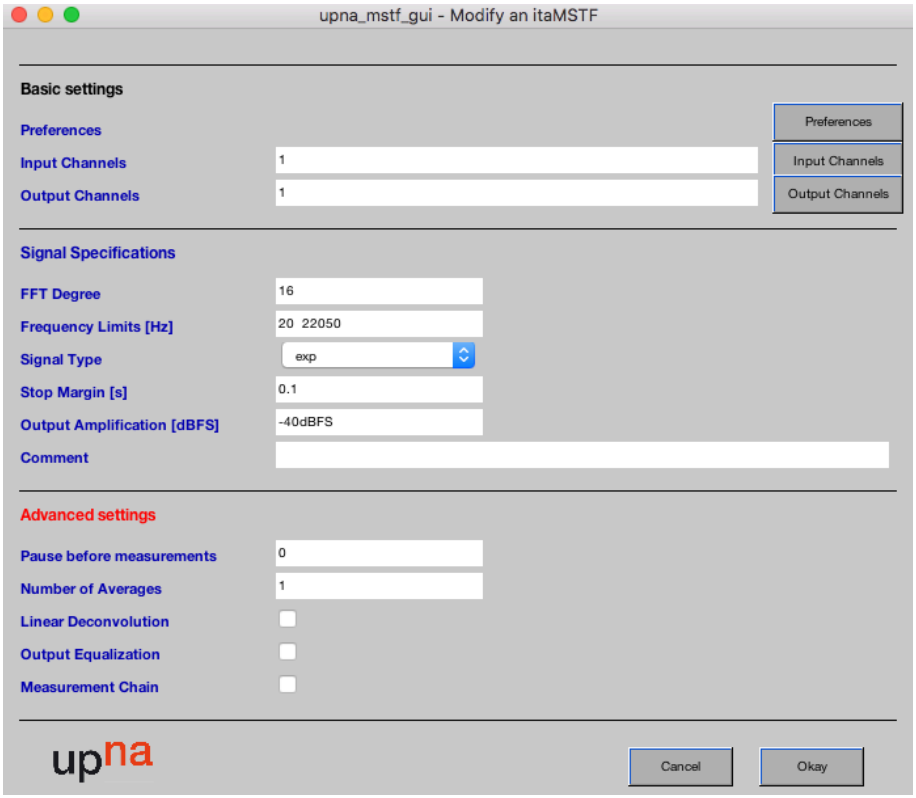

Ilustración 24: Ventana modificación parámetros de medida para itaMSTF

Concretamente la ventana que se muestra sería característica para la obtención de la función de transferencia de una sala. Como se puede observar el número de

características a modificar es sumamente amplio como para adecuarse a nuestros requerimientos.

Choose Measurement

La elección de la medida nos permitirá en todo momento estar pendientes de la medida a manipular. Esta opción está estrechamente relacionada con los apartados de cargar y guardar preferencias de medida.

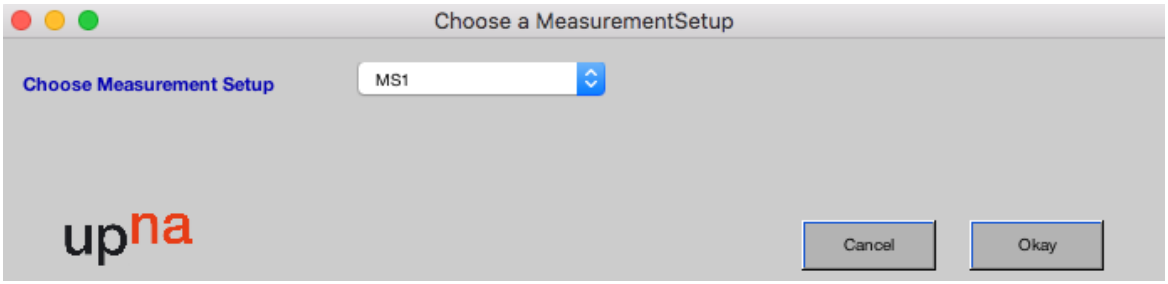

Ilustración 25: Ventana selección objeto de medida

A través de un desplegable se puede seleccionar la medida a manipular en cada momento.

Edit Measurement

Editar medida nos permite modificar los parámetros de una medida ya ejecutada previamente. Se nos mostrará la misma ventana (ilustración 24) vista anteriormente para la realización de una nueva medida.

Load Measurement Prefs

Carga un archivo **.mat* con los valores de medida que este archivo contenga, anteriormente se pueden guardar como veremos a continuación mediante la opción *Save Measurement Prefs*.

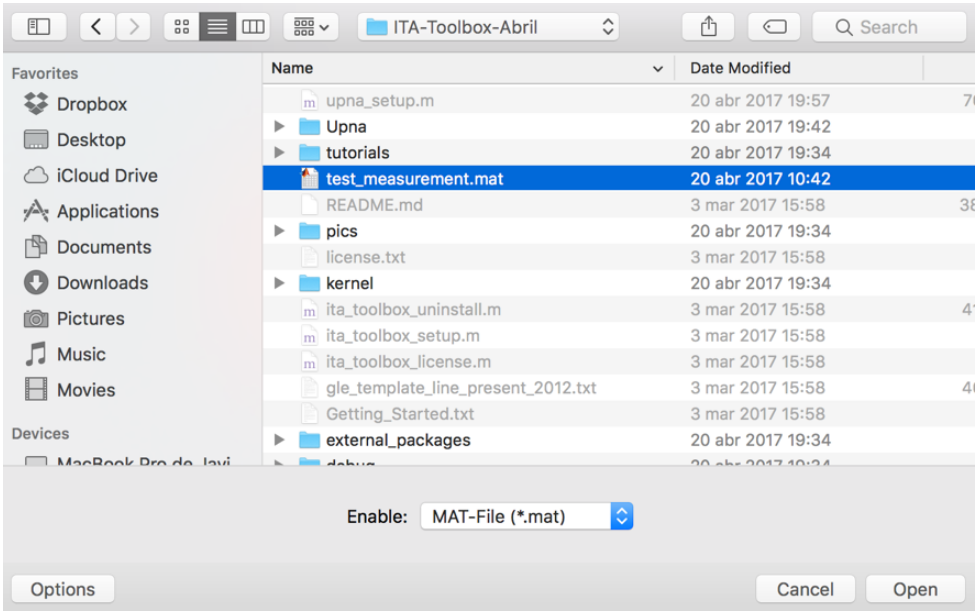

Ilustración 26: Ventana carga preferencias

Save Measurement Prefs

Nos almacena en una variable característica de MATLAB (**.mat*) las preferencias de una medida para poder emplearlas en un futuro. Cuando realicemos la exportación de esta variable a un fichero se deberá especificar el nombre y ubicación donde deseemos almacenar estas preferencias.

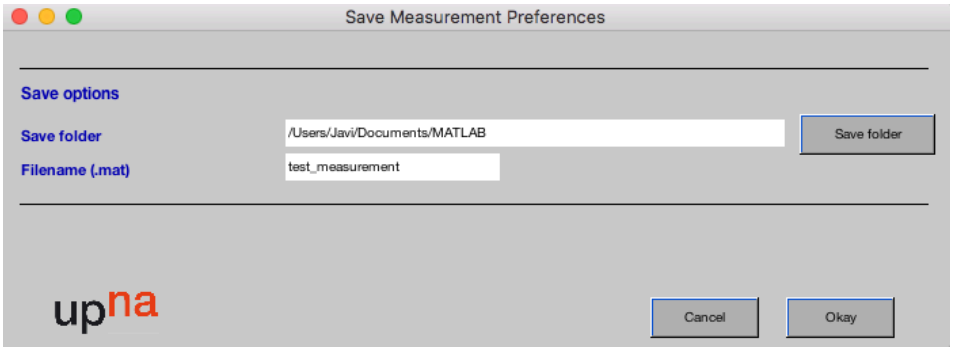

Ilustración 27: Ventana guardar preferencias

Calibrate Measurement

Para la correcta realización de las medidas era necesario crear un proceso de calibración, tras analizar el código de la ITA referente a la calibración se decidió realizar un proceso más simple pero eficiente igualmente.

Ilustración 28: Ventana emergente avisando que no se ha realizado la calibración

```
function upna_calibracion;

% <GUAIRTE-Toolbox>

f = helpdlg('Coloque el calibrado a 94dB sobre el micrófono', 'Calibrate Process');
% disp('This prints immediately');
drawnow     % Necessary to print the message
waitfor(f);
% disp('This prints after you close the warning dialog');

MS_SNR = itaMSRecord('fftDegree',16,'inputChannels',1);
Noise_r = MS_SNR.run;

Noise_m = ita_spk2frequencybands(Noise_r,'bandsperoctave',1,'freqRange',[900 1100]);
Calibracion.Level__1k = 20*log10(Noise_m.freq);

Calibracion.Compensacion = 94 - Calibracion.Level__1k;

clear f;

assignin('base', 'Calibracion', Calibracion)

end
```

El proceso consiste en grabar el nivel que recibimos en el micrófono colocando en este un calibrador con una señal de 94dB a 1kHz de frecuencia. Una vez realizada la grabación y el filtrado en la banda en torno a 1kHz se procesa la posible compensación necesaria de los decibelios teóricos frente a los obtenidos. Este valor de compensación se almacena en una variable de tipo *struct* en la que encontramos internamente el nivel medido a frecuencia 1kHz en dBs y el valor de compensación también expresado en dBs.

Una vez realizada la calibración no se mostrará el mensaje de advertencia indicado anteriormente. También es importante destacar que no debe manipular ningún potenciómetro de ganancia de la tarjeta de sonido, en caso contrario será necesario volver a realizar el proceso de calibración.

Run Measurement

Una vez hemos configuradas las preferencias convenientes según el tipo de medida y tras una previa calibración de los diferentes componentes que componen el sistema de medida procedemos a ejecutar el proceso.

Cuando finalice la ejecución se mostrará en la ventana principal un gráfico con la respuesta en frecuencia de la captura de audio realizada. Es aquí momento en el que cobran sentido la infinidad de funciones que incluye la ITA para la manipulación de señales de audio.

Levels

La aplicación niveles (*levels*) tiene una estrecha relación con los procesos de medida vistos en la aplicación anterior, concretamente utilizaremos un proceso de medida de tipo *itaMSRecord* y posteriormente se realizará su respectiva representación en bandas de octava. Será necesaria la realización de una calibración del sistema previa al proceso de medida para obtener resultados satisfactorios.

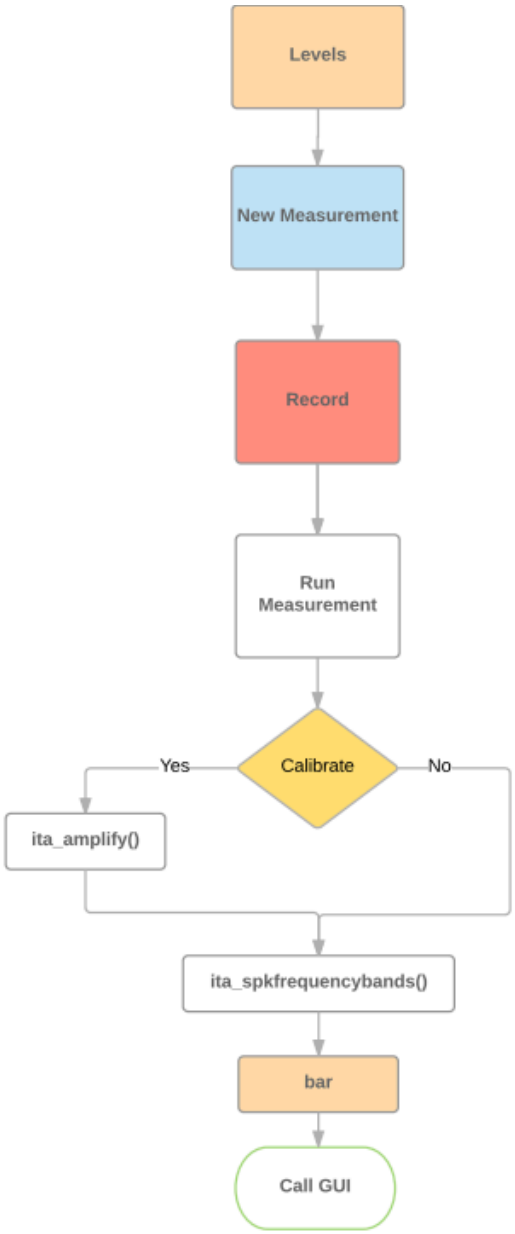

Diagrama de flujo 3: Aplicación Levels

Centrándonos ya en la programación de esta aplicación en primer lugar será necesaria una ventana donde se requiera introducir la duración de la grabación, así como los correspondientes canales de entrada seleccionados. Con estas preferencias se dispone de la siguiente ventana:

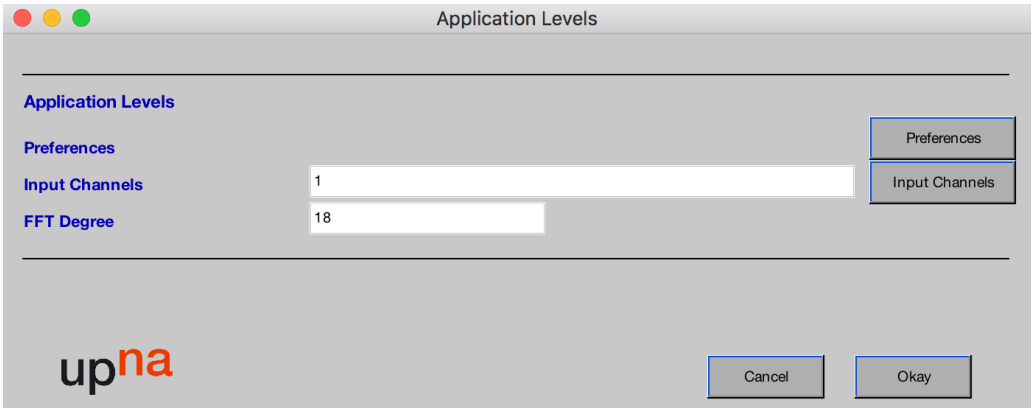

Ilustración 29: Ventana Application Levels

Como objeto principal se generará un *itaMSRecord* a partir del cual grabaremos el ruido de fondo de una sala, con la duración especificada por el usuario en el grado de la FFT y posteriormente esta grabación será almacenada en un objeto de tipo *itaAudio*.

```
MS_SNR = itaMSRecord('fftDegree', pList{3}, 'inputChannels', pList{1});
levels = MS_SNR.run;

if busqueda
    Calibracion = evalin('base', 'Calibracion');
    levels = ita_amplify(levels,Calibracion.Compensacion, 'dB');
end
```

En caso de realizar la correspondiente calibración que se ha comentado al inicio de la explicación se realizará la corrección de la medida a través de la función creada por parte de la ITA denominada *ita_amplify().*

Finalmente, procederemos a crear la función de representación de los niveles en la ventana principal en forma de barras, dividiendo mediante la función *ita_spk2frequencybands()* el objeto *itaAudio* en bandas frecuenciales para su correcta representación.

```
function upna_menucallback_Levels(hObject, event)

% <GUIARTE-Toolbox>

fgh = ita_guisupport_getParentFigure(hObject);

levels = upna_levels();

result = ita_spk2frequencybands(levels);

upna_guisupport_updateGUI_bar(fgh, result);

end


function upna_guisupport_updateGUI_bar(fgh, result)
% plot audio in figure

data =  getappdata(fgh,'audioObj');

clf(fgh, 'reset')

bar(result,'figure_handle',fgh);

combined_menu('handle',fgh,'type',data);

end
```

Como resultado final del proceso y teniendo en cuenta el código fuente anteriormente mostrado obtendríamos:

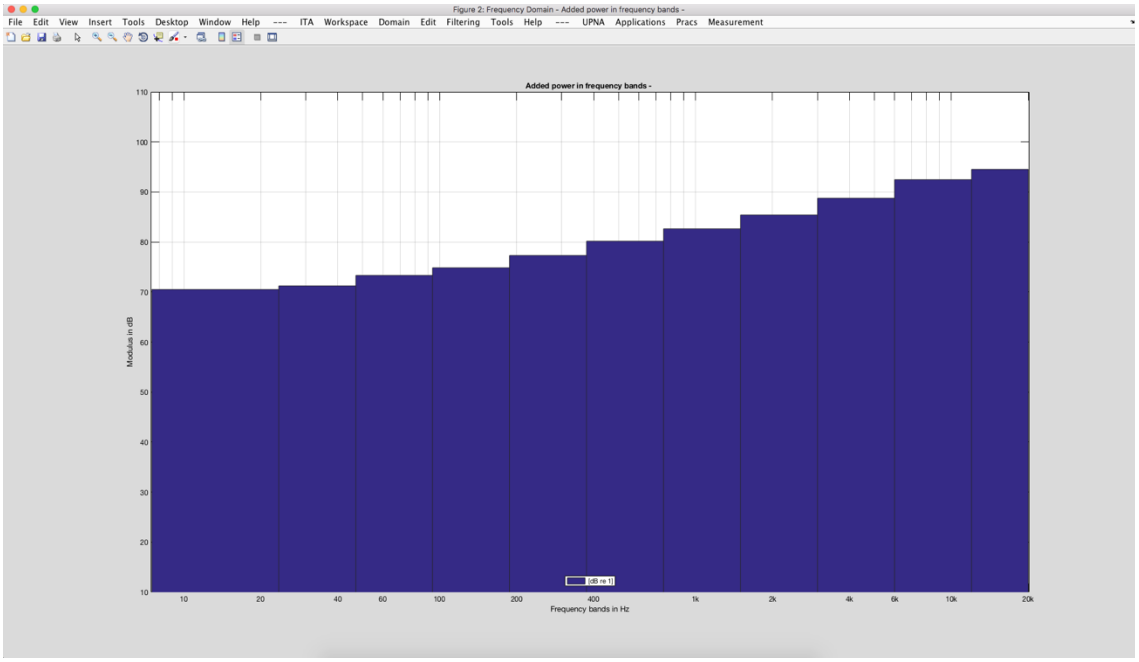

Ilustración 30: GUI representando los niveles de la señal grabada en bandas de octava

STI-Full Indirect

La siguiente aplicación sigue en relación la evaluación acústica característica de una sala, en este caso programaremos una *app* para el cálculo de la inteligibilidad. La evaluación se realizará a través del método indirecto, es decir, a través de los niveles en las bandas de 125Hz a 8kHz y la relación señal a ruido en el mismo rango frecuencial.

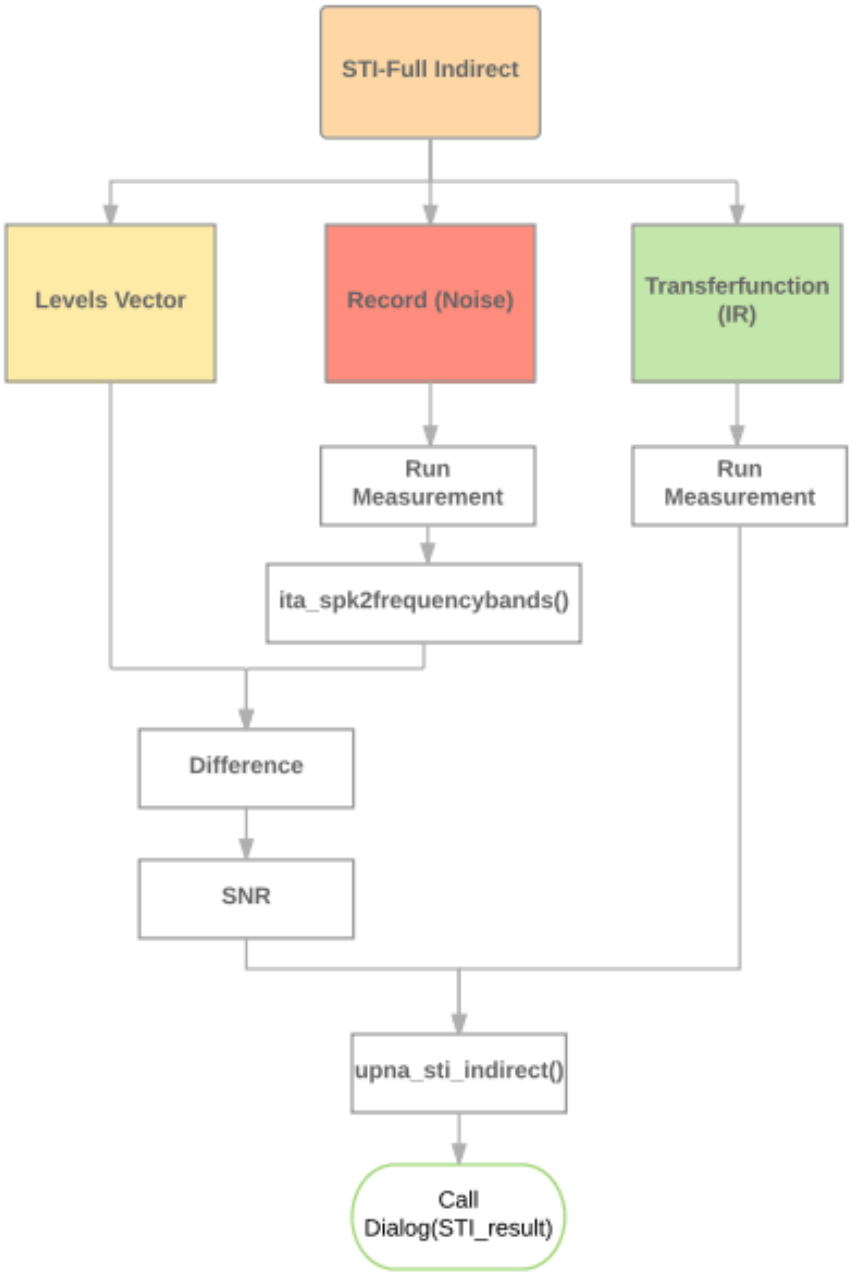

Diagrama de flujo 4: Aplicación STI-Full Indirect

Siguiendo como referencia alguna función implementada por la ITA en este caso nos percatamos de un script denominado *ita_speech_transmission().* Tras su evaluación consideramos realizar algunas modificaciones al mismo creando una nueva función que

conforme a nuestras bases pasó a llamarse *upna_sti_indirect().* Los parámetros de entrada necesarios para su correcto funcionamiento son:

- **IR:** Respuesta al impulso medida del recinto en el que se va a evaluar la inteligibilidad.
- **Niveles:** Vector de niveles en dBs correspondientes a las bandas de octava entre 125Hz y 8kHZ.
- **Relación Señal Ruido (SNR):** Vector de SNR en dBs correspondiente a las bandas de octava entre 125Hz y 8kHz.

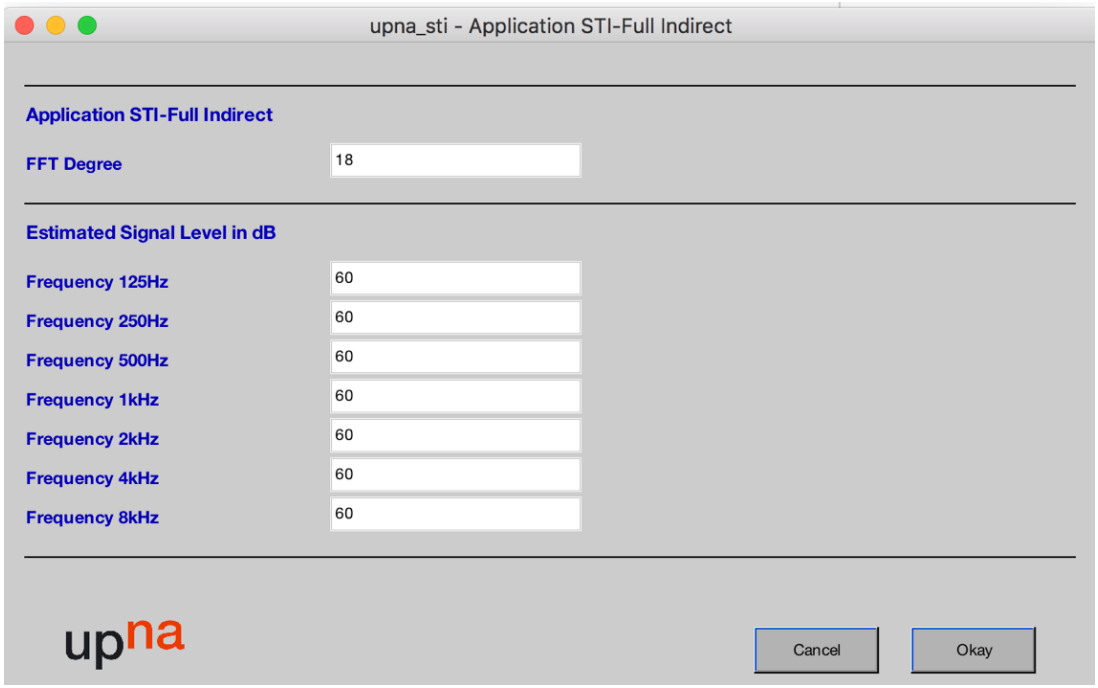

Ilustración 31: Ventana inserción de parámetros STI-Full Indirect

Para la obtención de estos parámetros de entrada en la función indicada será necesario realizar la siguiente implementación:

```
levels = [pList{2} pList{3} pList{4} pList{5} pList{6} pList{7} pList{8}];

MS_IR = itaMSTF('fftDegree', pList{1}, 'inputChannels', 1, 'outputChannels',1);
ir = MS_IR.run;

MS_Noise = itaMSRecord('fftDegree', pList{1}, 'inputChannels', 1);
Noise = MS_Noise.run;

Noise = ita_spk2frequencybands(Noise,'bandsperoctave',1,'freqRange',[125 8000]);
```

Una vez completado este procedimiento procedemos al cálculo de la SNR y la ejecución de la función anteriormente creada para el STI.

```
Noise_dB = 20*log10(Noise.freqData)

SNR = levels - Noise_dB;
STI_result = upna_sti_indirect(ir, 'levels', levels, 'SNR', SNR);
```

Introduciendo en la función la respuesta al impulso, vector de niveles introducido manualmente a través de la ventana inicial por parte del usuario y el vector SNR calculado anteriormente mediante las líneas código adjuntas.

Para finalizar se implementó a través de un cuadro emergente la impresión del resultado de inteligibilidad obtenido.

```
d = dialog('Position',[700 500 250 150],'Name','STI-Full Indirect')

txt = uicontrol('Parent',d,...
           'Style','text',...
           'FontSize',18,...
           'Position',[20 80 210 40],...
           'String',STI_result);

btn = uicontrol('Parent',d,...
           'Position',[85 20 70 25],...
           'String','Close',...
           'Callback','delete(gcf)');
```

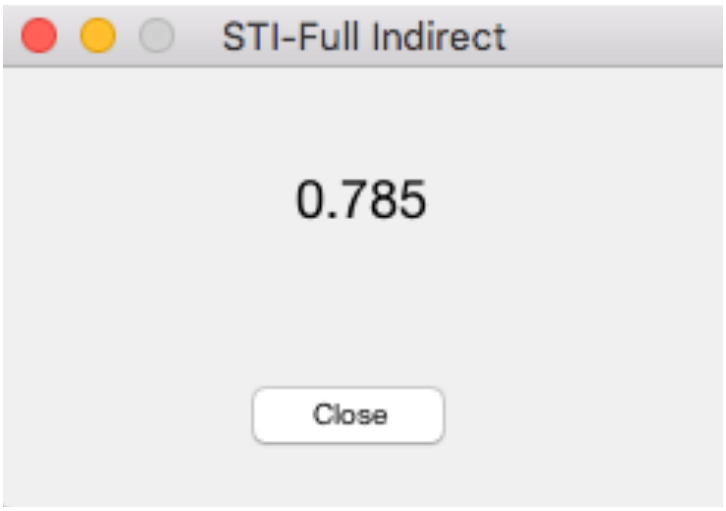

Ilustración 32: Ventana diálogo mostrando el valor del STI

Capítulo 3: Generación de prácticas y análisis de funcionalidades

Introducción

En este capítulo, siguiendo con la evaluación de posibilidades que nuestra herramienta implementada nos ofrece, vamos a realizar pequeñas funciones a modo de ejemplos prácticos a través de las cuales nos familiarizaremos aún más con el entorno de programación definido.

Además, estas prácticas serán de gran utilidad para los alumnos de grado de manera que puedan visualizar y contemplar los conceptos teóricos explicados en clase.

Prácticas

Práctica 1: Evaluación del tiempo de reverberación (T20)

La primera práctica propuesta consiste en la realización de varias medidas de respuesta de una sala en diferentes puntos de la misma y su posterior análisis a través del tiempo de reverberación. La disposición fuente-receptor sería similar a la que se adjunta en la siguiente ilustración:

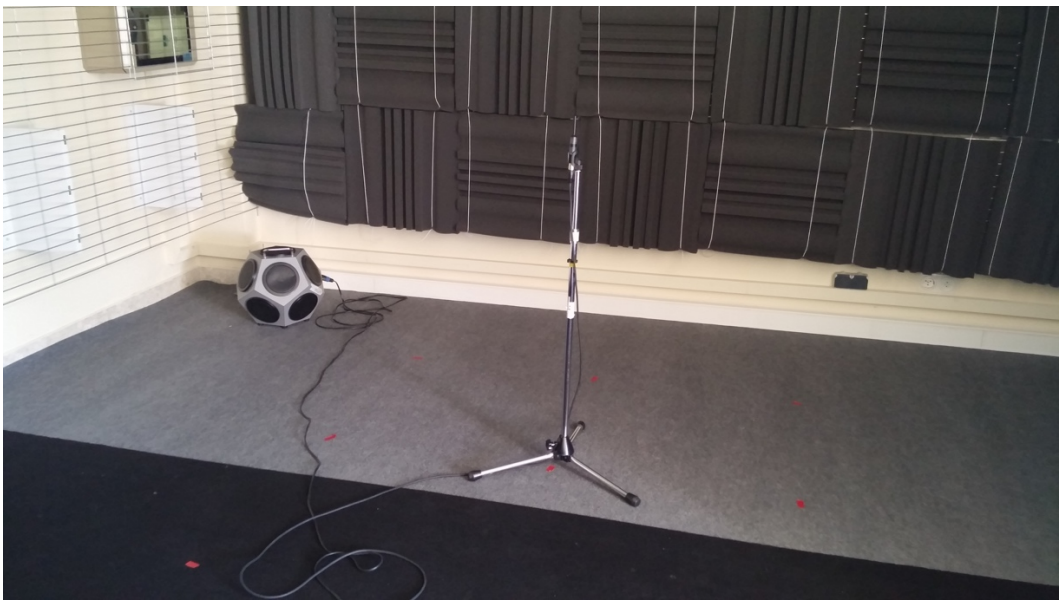

Ilustración 33: Fuente dodecaédrica y micrófono situado en el primer punto de medida

El modelo de ventana a programar requerirá de la duración de la medida y el número de receptores. Para ello, beneficiándonos de las funciones implementadas y comentadas con anterioridad implementaremos el siguiente script.

```
% <GUIARTE-Toolbox>

    ele = 1;
    pList{ele}.description = 'FFT Degree';
    pList{ele}.helptext    = 'This is the itaAudio Object for amplification or attenuation';
    pList{ele}.datatype    = 'int';
    pList{ele}.default     = '18';

    ele = 2;
    pList{ele}.description = 'Numero receptores';
    pList{ele}.helptext    = 'Factor can be e.g. ''0dB'' or ''1+5j'' or ''20 Pa'' ' ;
    pList{ele}.datatype    = 'int';
    pList{ele}.default     = '5';

    ele = 3;
    pList{ele}.datatype    = 'line';

    %call gui
    pList = upna_parametric_GUI(pList,[mfilename ' - Prac1 T20 with Receptors']);
```

Como resultado se muestra la siguiente ventana por pantalla.

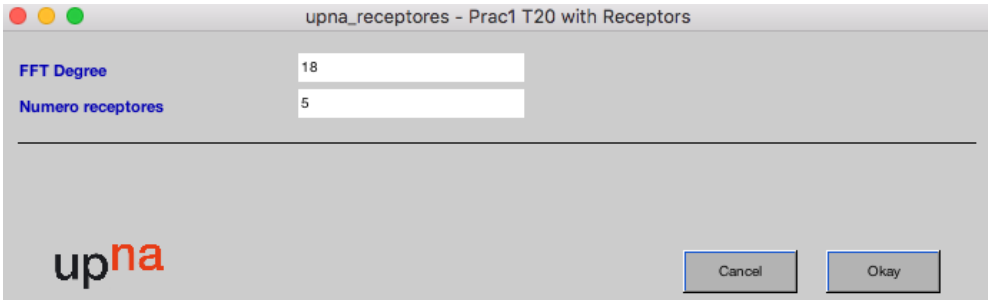

Ilustración 34: Ventana Prac1 evaluación del tiempo de reverberación (T20)

Una vez seleccionada la duración y el número de receptores procederemos a realizar las correspondientes medidas. Es en este momento cuando nos surge el siguiente problema, necesitamos parar la ejecución del script tras cada medida de tal forma que podamos cambiar de posición el receptor. A través de la inclusión del método *waitfor()* de MATLAB.

```
f = warndlg('Medida realizada. Coloque el siguiente receptor','Warning!!');
waitfor(f);
```

De esta forma somos capaces de detener el script y hasta que no confirmemos que hemos realizado el correspondiente cambio del receptor no procederá a ejecutar la siguiente medida. El funcionamiento completo del script se muestra en el siguiente diagrama de flujo:

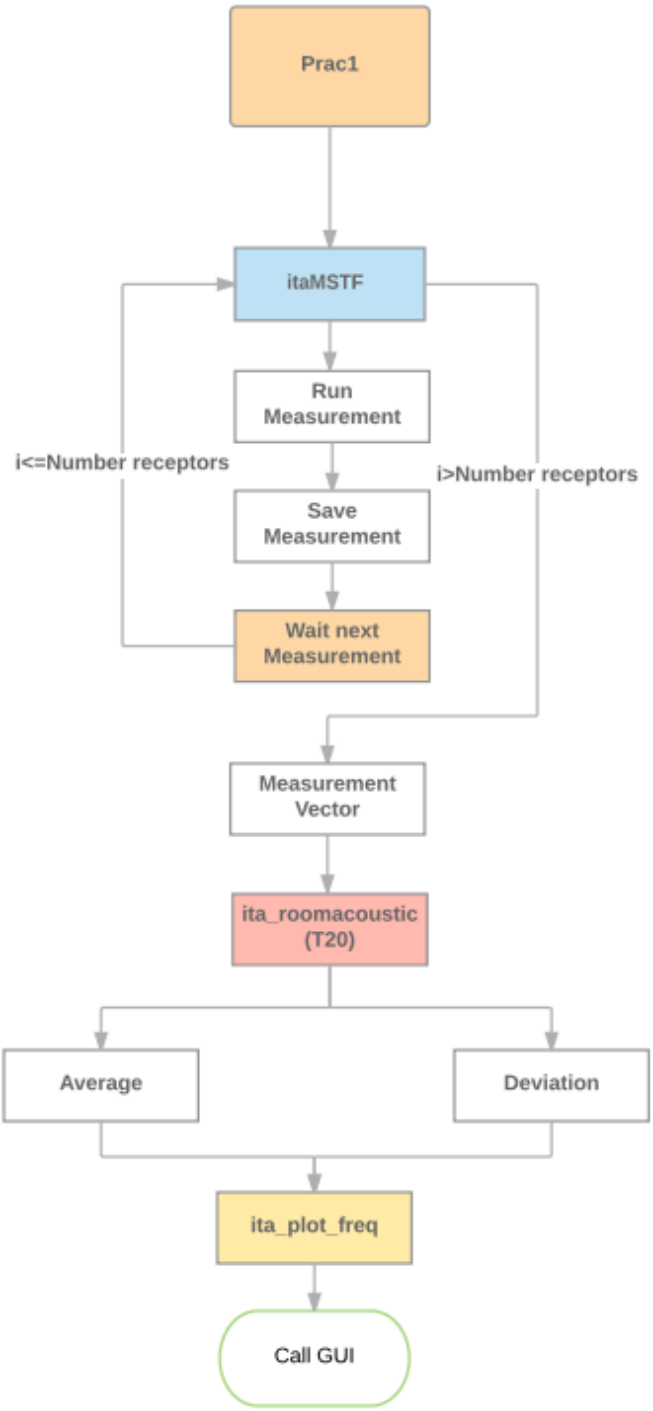

Diagrama de flujo 5: Prac1 evaluación del tiempo de reverberación (T20)

Tras completar el número de medidas deseadas en los diferentes puntos fijados, procederemos a el cálculo del tiempo de reverberación (T20) haciendo uso de la función *ita_roomacoustic().* Calculando el tiempo de reverberación para cada una de las medidas de *IR* realizadas y seguidamente realizaremos la media de los tiempos de reverberación por bandas y su respectiva desviación representándolo por pantalla.

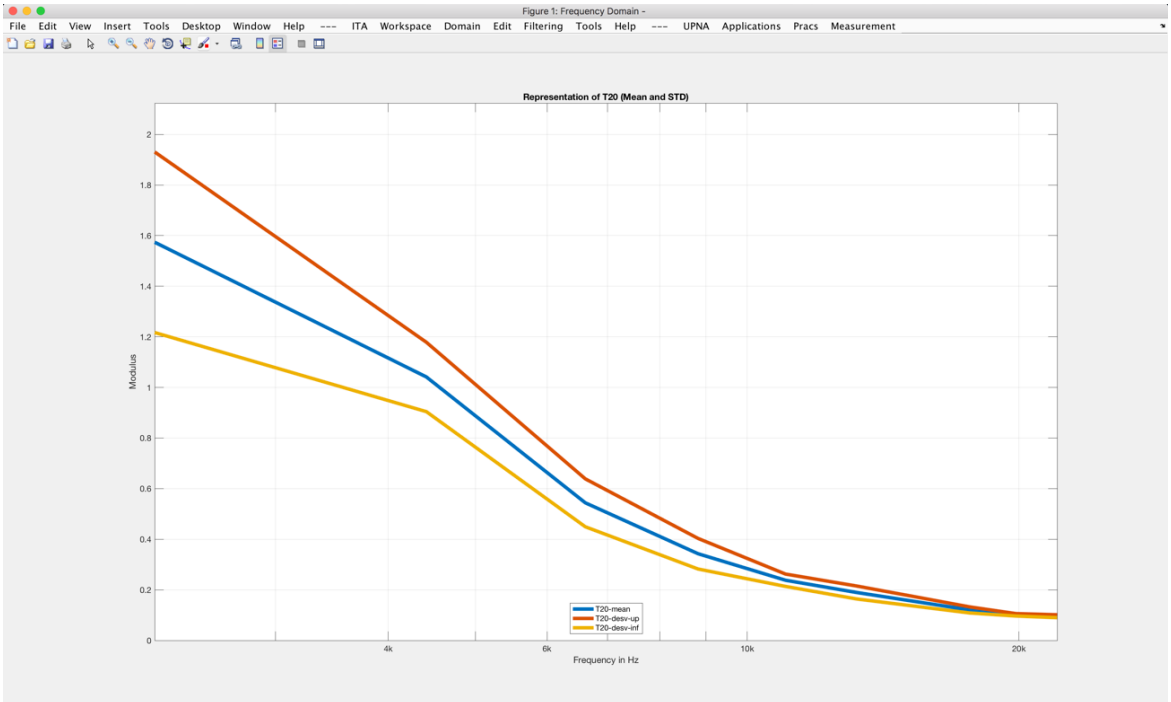

Ilustración 35: Representación de la media y desviación del tiempo de reverberación

La línea azul representa la media del tiempo de reverberación y la línea rojo y amarillas representan la media más la desviación y la media menos la desviación, respectivamente.

Práctica 2: Evaluación de la respuesta frecuencial de auriculares

Como segunda propuesta se consideró interesante sacar partido a la cabeza binaural disponible, de tal forma que programando una función pudiésemos ser capaces de evaluar las diferencias, en cuanto a respuesta frecuencial se refiere, de dos modelos de auriculares. Definiendo los parámetros que debería contener la ventana referente a este apartado se consideraron como entradas necesarias:

- **Preferencias:** accedemos a la ventana de *upna_preferences()*.
- **Canales de entrada:** necesarios 2 canales de entrada para el correcto funcionamiento.
- **Canales de salida:** necesarios 2 canales de salida para el correcto funcionamiento.
- **Duración del proceso:** duración de la medida a realizar
- **Auricular 1:** nombre del modelo de auricular 1 a evaluar

- **Auricular 2:** nombre del modelo de auricular 2 a evaluar

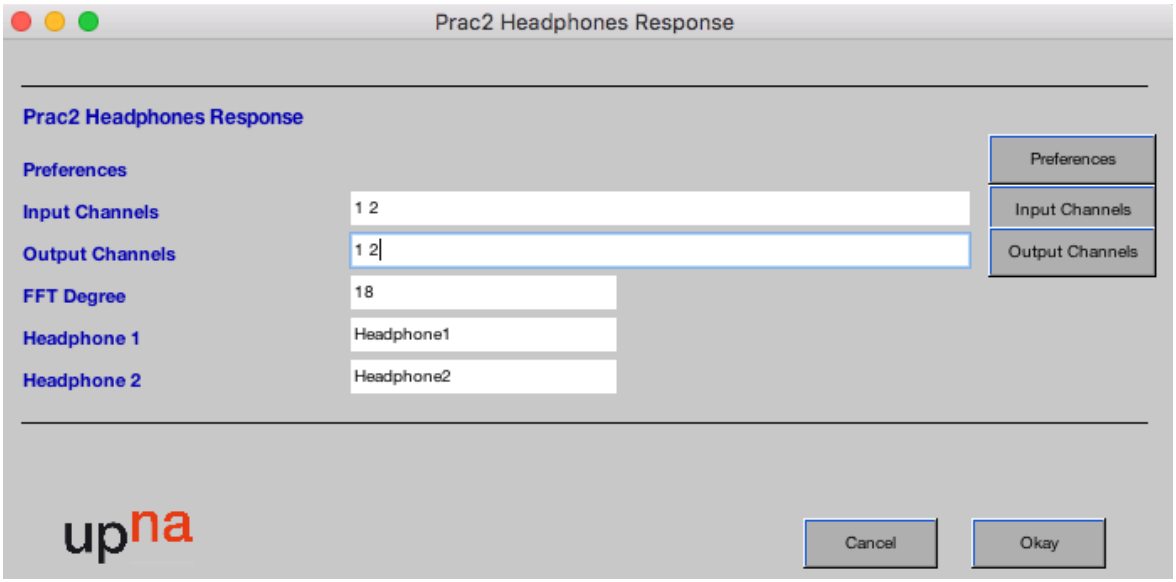

Ilustración 36: Ventana Prac2 evaluación de la respuesta frecuencia de auriculares

Tras la entrada de parámetros en la ventana anterior se procede a realizar las medidas correspondientes a través de los objetos de la clase *itaMSTF*. Estos objetos serán almacenados en el *workspace* con objeto de poder exportarlos para posteriores evaluaciones o comparación con otros equipos.

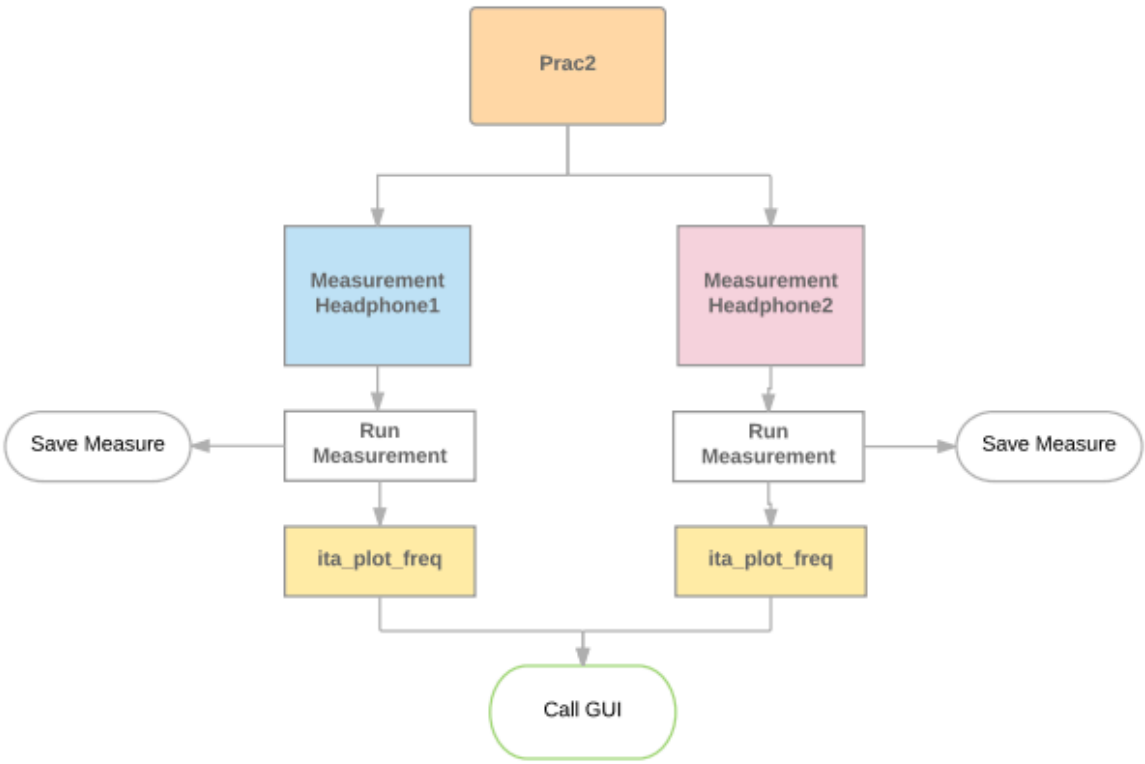

Diagrama de flujo 6: Prac2 evaluación de la respuesta frecuencial de auriculares

```
pList = upna_parametric_GUI(pList,'Prac2 Headphones Response');


if isempty(pList)
    return;
end

    Headphone1 = itaMSTF('fftDegree', pList{3}, 'inputChannels', pList{1}, 'outputChannels', pList{2})
    Headphone1_result = Headphone1.run;
    Headphone1_result.comment = pList{4};
    assignin('base', pList{4}, Headphone1_result);

    Headphone2 = itaMSTF('fftDegree', pList{3}, 'inputChannels', pList{1}, 'outputChannels', pList{2})
    Headphone2_result = Headphone2.run;
    Headphone2_result.comment = pList{5};
    assignin('base', pList{5}, Headphone2_result);
```

Siguiendo la notación indicada por la ITA creamos dos objetos *itaMSTF* con las respectivas características que el usuario ha seleccionado en la ventana inicial. Dichas características son almacenadas en un listado denominado concretamente *pList*, siendo *pList{3}* el grado de la FFT, *pList{1}* los canales seleccionados de entrada y *pList{2}* los canales seleccionados de salida. Tras definir las medidas se ejecutan mediante la instrucción *.run* y se almacena en un objeto el resultado con el respectivo nombre.

Ilustración 37: Cabeza binaural con auriculares AKG Studio

Tras la realización de las dos medidas procedemos a la representación gráfica de los resultados. Para ello, se consideró interesante mostrar ambos gráficos independientemente dividendo la pantalla en dos como se ha realizado en el caso de la aplicación referente a parámetros acústicos.

Creamos una nueva función de tipo GUI en la que como se ha indicado se precisa de la división de la ventana principal de nuestra herramienta, tal y como vemos a continuación a través de la función *upna_guisupport_updateGUI_freq()*.

```
function upna_guisupport_updateGUI_freq(fgh, h1, h2)

% plot audio in figure
currentDomain = getappdata(fgh,'ita_domain');
data =  getappdata(fgh);

clf(fgh, 'reset')

if isempty(currentDomain)
    error('')
end

subplot(2,1,1);
    ita_plot_freq(h1,'figure_handle',fgh);
    title(h1.comment);
    legend('L signal','R siganl');


subplot(2,1,2);
    ita_plot_freq(h2,'figure_handle',fgh);
    title(h2.comment);
    legend('L signal','R siganl');

combined_menu('handle',fgh);

end
```

Obteniendo como resultado la siguiente representación en la que se puede ver una comparación entre unos auriculares profesionales frente a otros de una calidad claramente inferior.

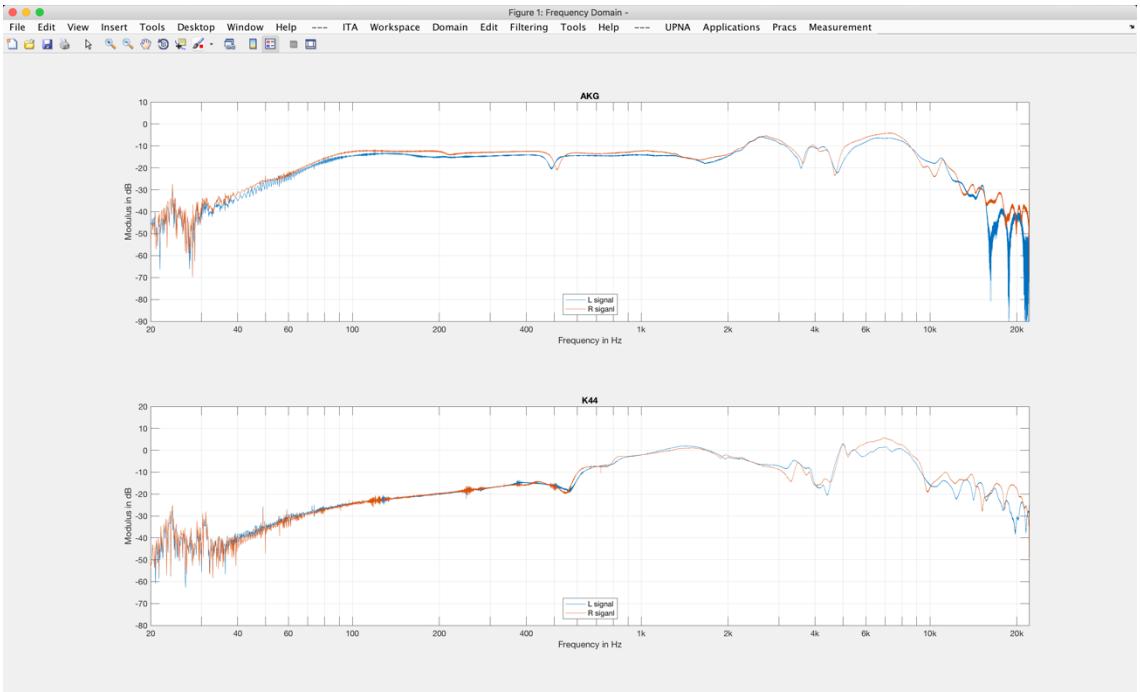

Ilustración 38: Representación frecuencial auricular 1 (AKG) frente a auricular 2 (K44)

Práctica 3: Evaluación de prestaciones de protectores auditivos frente al ruido

Finalmente, manteniendo como protagonista la cabeza binaural se procedió a la evaluación de unos protectores auditivos frente al ruido. De este modo introducimos la funcionalidad que incorpora la ITA para la generación de señales, denominada *ita_generate()*. Para este ejemplo utilizaremos concretamente ruido rosa (mismo nivel en todas las bandas) que será emitido posteriormente a través de una fuente.

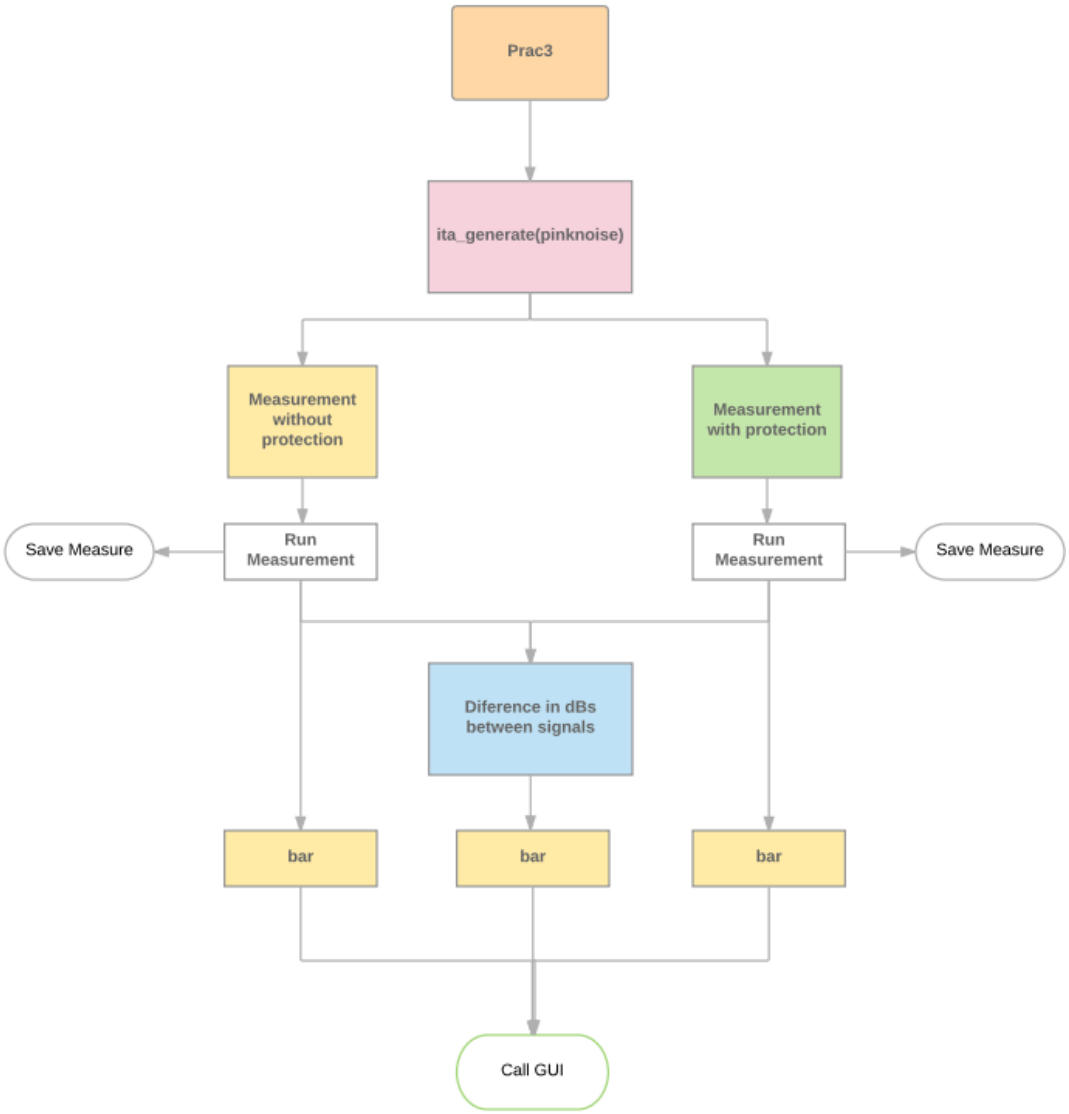

Diagrama de flujo 7: Prac3 evaluación de prestaciones de protectores auditivos frente al ruido

```
noise = ita_generate('pinknoise',1,44100, pList{3})
```

El proceso consistirá en realizar una comparación de los niveles obtenidos por la cabeza binaural cuando no tenemos protección frente a los niveles medidos en el caso

de tener protección. Para su ejecución emplearemos un objeto *itaMSPlaybackRecord*, mediante el cual somos capaces de emitir una señal cualquiera, elegida por el usuario, y grabar simultáneamente la respuesta que obtenemos en los receptores.

La ventana programada para esta práctica es sencilla y similar a la empleada en la práctica anterior, por lo que no entraremos en detalle. Únicamente mencionar que es necesario el empleo de dos canales de entrada puesto que se trata de una cabeza binaural (imagen stereo).

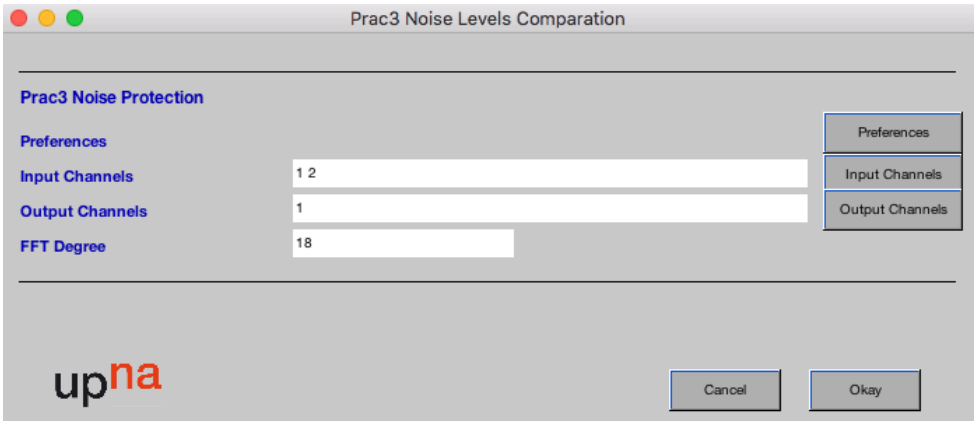

Ilustración 39: Ventana prac3 evaluación de prestaciones de protectores auditivos frente al ruido

Tras la correcta elección de los valores elegidos por parte del usuario se procederá a realizar la primera medida sin ningún tipo de protección auditiva. La disposición de la cabeza binaural será la mostrada en la ilustración 40.

Ilustración 40: Cabeza binaural sin protección

Una vez finalizada la primera medida se nos mostrará un mensaje para que coloquemos en la cabeza binaural los auriculares de protección y una vez colocados aceptaremos el mensaje emergente y se realizará la siguiente medida.

Ilustración 41: Cabeza binaural con protección

Tras la ejecución de la segunda medida se procederá a almacenar en el entorno de trabajo de MATLAB ambas grabaciones en forma de objetos *itaAudio* con el objetivo de poder guardar estas medidas para futuras comparativas. Seguidamente se evaluará la diferencia de niveles entre la medida con protección y la de sin protección.

```
PlayRecord1 = itaMSPlaybackRecord('fftDegree', pList{3}, 'inputChannels', pList{1}, 'outputChannels', ...
                                  pList{2}, 'excitation', noise);
PlayRecord_result1 = PlayRecord1.run;
PlayRecord_result1.comment = 'Without Protection';
assignin('base', 'WithoutProtection', PlayRecord_result1);

PlayRecord2 = itaMSPlaybackRecord('fftDegree', pList{3}, 'inputChannels', pList{1}, 'outputChannels', ...
                                  pList{2},  'excitation', noise)
PlayRecord_result2 = PlayRecord2.run;
PlayRecord_result2.comment = 'With Protection';
assignin('base', 'WithProtection', PlayRecord_result2);
```

Previamente a evaluar debemos destacar que se han amplificado 100dBs ambas señales para representar posteriormente los niveles en un rango positivo. Tras esta amplificación se realizará la división de las medidas en bandas frecuenciales y se efectuará la diferencia entre las medidas.

```
Noise1 = evalin('base','RuidoRosaRec1');
Noise2 = evalin('base','RuidoRosaRec2');
Noise1 = ita_amplify(Noise1, 100, 'dB');
Noise2 = ita_amplify(Noise2, 100, 'dB');

result1 = ita_spk2frequencybands(Noise1);
result2 = ita_spk2frequencybands(Noise2);
```

```
diferencia = 20*log10(result1.freqData)- 20*log10(result2.freqData);

diferencia = diferencia/20;
diferencia = 10.^diferencia;

result1.freqData = diferencia;

subplot(2,1,1);
    bar(result1,'figure_handle',fgh);
    title('Difference Noise Level');
    legend('L signal','R signal');

subplot(2,2,3);
    bar(Noise1,'figure_handle',fgh);
    title('Noise Level without protection');
    legend('L signal','R signal');

subplot(2,2,4);
    bar(Noise2,'figure_handle',fgh);
    title('Noise Level with protection');
    legend('L signal','R signal');


combined_menu('handle',fgh);
```

Finalmente se hará referencia al apartado de representación de niveles en el que encontraremos una disposición de tres gráficos (diferencia de niveles, medida sin protección y medida con protección).

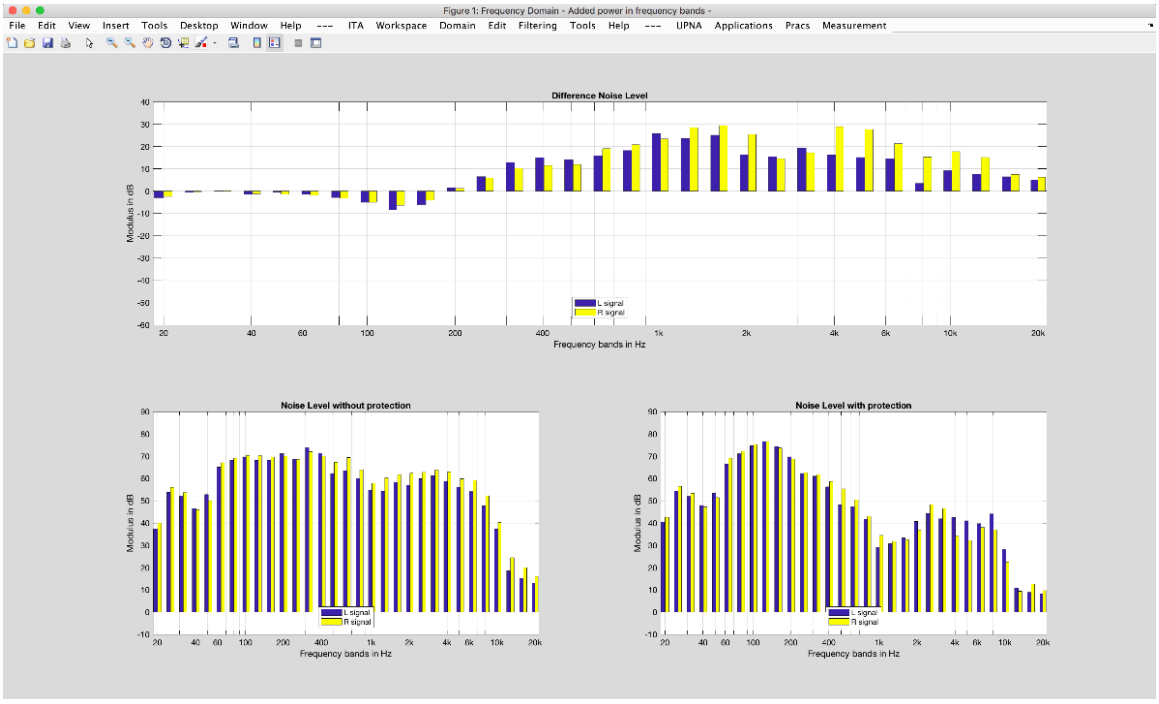

Ilustración 42: Representación niveles con y sin protección auditiva y diferencia entre niveles

Como curiosidad apreciamos a baja frecuencia valores negativos de nivel, este fenómeno podría ser causado debido al efecto de concha acústica procedente de los protectores auditivos. La concavidad de estos puede ser el causante del denominado efecto reforzando la energía acústica a bajas frecuencias.

Capítulo 4: Conclusiones y líneas futuras

Conclusiones

El objetivo principal de este trabajo de fin de grado era analizar las posibles herramientas *open source* activas en la actualidad, el análisis de las funcionalidades integradas y la selección de un software para su posterior documentación y desarrollo. Todo ello siempre con el fin de garantizar y fomentar su uso tanto por parte del equipo docente de la universidad, como del alumnado en tareas relacionadas con el estudio acústico.

A pesar de la amplia gama de aplicaciones extendidas en la red, fue de gran interés una *toolbox* creada por la *RTWH Aachen University* mediante la programación en MATLAB. Tras el correspondiente análisis se procedió a confeccionar un listado de mejoras con el fin de sustentar las necesidades del equipo universitario. Se presentaron los siguientes puntos:

- Creación de un menú simplificado e integrado con la ITA-Toolbox.
- Integración bidireccional entre ambas aplicaciones.
- Creación de aplicaciones propias empleando las beneficiosas utilidades que el motor de la ITA integra.
- Creación de prácticas a modo de ejemplos para la futura implementación de funcionalidades

Las conclusiones alcanzadas tras la realización del proyecto son las siguientes:

- GUIARTE-Toolbox es una herramienta válida para la realización de medidas y evaluación acústica de recintos, permitiendo tras un análisis previo, la comprensión del funcionamiento y la comunicación entre los objetos de audio y la interfaz gráfica.
- La aplicación desarrollada en el entono de MATLAB, permite la posibilidad de realizar procesos de adquisición y generación de audio de manera simultánea a través de tarjetas de sonido sin ningún tipo de latencia o retardo.
- La ITA-Toolbox se encuentra en continuo desarrollo por parte de los programadores de la universidad de RTW Aachen. Además, recientemente se ha incluido en la plataforma *GitLab*, propiciándose de este modo una comunicación directa con sus creadores para la posible corrección de ciertas implementaciones.

Finalmente, tras el reconocimiento del propósito general de la aplicación, se van a introducir las mejoras que debido a limitaciones de tiempo no han sido implementadas de manera satisfactorio y, por tanto, no incluidas en la aplicación, resultado como posibles futuros trabajos.

Líneas futuras

Teniendo en cuenta uno de los propósitos principales por los que se ha inicializado la creación de esta *toolbox* se pretende que esta herramienta evolucione notoriamente en los próximos años. Beneficiándose de sus funcionalidades tanto el personal docente, en la investigación, como el alumnado para su correcto aprendizaje en lo que a acústica se refiere.

Una vez realizada la programación base de esta *toolbox*, es posible plantear como futuro trabajo la integración de nuevas aplicaciones para la evaluación acústica siguiendo los protocolos preestablecidos para la consiguiente fluidez de la aplicación. Sería interesante que dichas aplicaciones siguieran los métodos de ejecución impuestos en las normas ISO.

Por otra parte, resultaría interesante la generación de protocolos de medida y representación de señales de audio en tiempo real mediante el empleo de la *toolbox* propia de MATLAB denominada *Data Adquisition*, permitiendo así la posible programación de esta propuesta.

REFERENCIAS

[1] Institute of Technical Acoustics of the RWTH Aachen University. (2010). Recuperado de *http://www.ita-toolbox.org/*

[2] ITA-Toolbox -- An Open Source MATLAB Toolbox for Acousticians. (2012). En *ResearchGate.* Recuperado de *https://www.researchgate.net/publication/233762454_ITA-Toolbox_--_An_Open_Source_MATLAB_Toolbox_for_Acousticians*

[3] MATLAB Documentation. (2017). En *Mathworks*. Recuperado *de https://es.mathworks.com/help/matlab/*